

Copyright  
by  
Cheryl Lynn Brabec  
2013

**The Thesis Committee for Cheryl Lynn Brabec  
Certifies that this is the approved version of the following thesis:**

**A shape primitive-based grasping strategy using visual object  
recognition in confined, hazardous environments**

**APPROVED BY  
SUPERVISING COMMITTEE:**

**Supervisor:**

\_\_\_\_\_  
Sheldon Landsberger

**Co-Supervisor:**

\_\_\_\_\_  
Mitchell Pryor

**A shape primitive-based grasping strategy using visual object  
recognition in confined, hazardous environments**

**by**

**Cheryl Lynn Brabec, B.A.**

**Thesis**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Master of Science in Engineering**

**The University of Texas at Austin**

**December 2013**

## **Acknowledgements**

Foremost, I would like to express my sincere gratitude to my advisors Dr. Sheldon Landsberger and Dr. Mitch Pryor. Dr. Pryor's patience and guidance during the writing process have been invaluable, and Dr. Landsberger's enduring support has left me eternally grateful.

I would also like to thank Los Alamos National Laboratory and the Nuclear Regulatory Commission for their financial support during the course of my Master's. Special thanks go out to LANL group AET-2 for their support and ever illuminating conversations.

Special acknowledgements go out to my dear family and friends who have always been there for me with kind words of encouragement. Last but not least, I'd like to thank FJB for always knowing the perfect way to cheer me on.

## **Abstract**

### **A shape primitive-based grasping strategy using visual object recognition in confined, hazardous environments**

Cheryl Lynn Brabec, M.S.E

The University of Texas at Austin, 2013

Supervisor: Sheldon Landsberger

Co-supervisor: Mitchell Pryor

Grasping can be a complicated process for robotics due to the replication of human fine motor skills and typically high degrees of freedom in robotic hands. Robotic hands that are underactuated provide a method by which grasps can be executed without the onerous task of calculating every fingertip placement. The general shape configuration modes available to underactuated hands lend themselves well to an approach of grasping by shape primitives, and especially so when applied to gloveboxes in the nuclear domain due to the finite number of objects anticipated and the safe assumption that objects in the set are rigid. Thus, the object set found in a glovebox can be categorized as a small set of primitives such as cylinders, cubes, and bowls/hemispheres, etc. These same assumptions can also be leveraged for reliable identification and pose estimation within a glovebox. This effort develops and simulates a simple, but robust and effective grasp planning algorithm for a 7DOF industrial robot and three fingered dexterous, but underactuated robotic hand. The proposed grasping

algorithm creates a grasp by generating a vector to the object from the base of the robot and manipulating that vector to be in a suitable starting location for a grasp. The grasp preshapes are selected to match shape primitives and are built-in to the Robotiq gripper used for algorithm demonstration purposes. If a grasp is found to be unsuitable via an inverse kinematics solution check, the algorithm procedurally generates additional grasps to try based on object geometry until a solution can be found or all possibilities are exhausted. The algorithm was tested and found capable of generating valid grasps for visually identified objects, and can recalculate grasps if one is found to be incompatible with the current kinematics of the robotic arm.

## Table of Contents

List of Tables .....	x
List of Figures .....	xi
Chapter 1 : Introduction .....	1
1.1 Human Hands.....	1
1.2 Anthropomorphic Robotic Hands .....	2
1.3 Nonanthropomorphic Robotic Hands .....	3
1.4 Grasp Planning.....	5
1.4.1 Grasp Planning Suggested Sequence .....	5
1.5 Environmental Constraints.....	6
1.5.1 Nuclear Considerations .....	7
1.6 Objectives .....	10
1.7 Structure of Thesis .....	11
Chapter 2 : Literature Review.....	12
2.1 Early Grasping Work .....	12
2.2 Closure Properties of Grasps .....	15
2.3 Shape Primitives .....	17
2.3.1 Miller, Knoop, Christensen, and Allen .....	18
2.3.2 Elkvall and Kragic .....	23
2.4 Summary .....	24
Chapter 3 : <i>A Priori</i> Data, Sensing data, and Robotic Operational Software.....	26
3.1 <i>A priori</i> information.....	26
3.2 Sensing Technologies .....	29
3.2.1 Tactile Feedback .....	29
3.2.2 Computer Vision.....	30
3.2.3 Force Feedback .....	35
3.3 Operational Software Libraries.....	35
3.3.1 OSCAR .....	36

3.3.2 ROS.....	38
3.4 Summary .....	40
Chapter 4 : Proposed Algorithm Solution.....	41
4.1 Grasp Planning Algorithm .....	41
4.1.1 Key Assumptions .....	41
4.1.2 Object Shape Categories .....	42
4.1.3 Kinematic Construction of Grasp .....	44
4.1.3.1 Bowl.....	47
4.1.3.2 Cylinder.....	49
4.1.3.3 Box.....	51
4.2 Algorithm Review.....	52
Chapter 5: Hardware, Demonstration, and Analysis .....	53
5.1 Hardware.....	53
5.1.1 Robotiq Gripper .....	54
5.1.2 Motoman Arm.....	59
5.1.3 Sensors .....	59
5.1.4 Hardware Summary .....	60
5.2 Software Framework.....	61
5.2.1 ROS.....	61
5.2.2 AX.....	63
5.3 Initial Simulation Setup .....	64
5.4 Canister .....	64
5.5 Bowl.....	65
5.6 Inverse Kinematic False Negatives.....	66
5.7 Inverse Kinematic False Positives .....	67
5.8 Canister Workspace Test .....	67
5.9 Summary .....	68
Chapter 6 : Conclusion.....	70
6.1 Summary .....	70



6.2 Future Work .....	73
6.2.1 Deployment Research .....	73
6.2.2 Expanded Dataset.....	74
6.2.3 Task Heuristics.....	75
6.2.4 Grasp Optimization.....	75
6.3 Concluding Comments.....	76
Appendix – Source Code and Usage .....	77
References.....	100

## **List of Tables**

Table 1.1: Challenges and benefits for grasping inside a glovebox .....	10
Table 2.1: Shape primitives and associated grasping strategies, adapted from [Miller et al., 2003] .....	21
Table 3.1: Transitional levels of autonomy .....	28
Table 4.1: Shape primitive breakdown for LANL dataset items .....	44

## List of Figures

Figure 1.1: Human adaptability in grasping [Massa et al., 2002] .....	2
Figure 1.2: Examples of dexterous hands .....	3
Figure 1.3: The BarrettHand in various stages of finger rotation [Barrett Technology Inc., 2011] .....	4
Figure 1.4: The Robotiq hand [Robotiq, 2011] .....	4
Figure 1.5: (a) Parallel jaw gripper [Robotiq, 2013], (b) Balloon and coffee ground gripper [Ju, 2010] .....	4
Figure 1.6: The interior of a glovebox [Thompson, 2002] .....	7
Figure 1.7: A hot cell and worker using telemanipulation [SCK CEN, 2013] .....	8
Figure 2.1: Early multifingered hand design [Hanafusa and Asada, 1977] .....	13
Figure 2.2: Simplified version of Cutkosky’s grasping taxonomy [Murakami et al., 2009] .....	14
Figure 2.3: Spherical, cylindrical, precision-tip, and hook pregrasp shapes for the BarrettHand [Miller et al., 2003] .....	19
Figure 2.4: A model of a coffee mug and its associated model from shape primitives [Miller et al., 2003] .....	20
Figure 2.5: Examples for grasp generation on the four shape primitive types. Balls represent starting positions for the center of the palm, and long arrows represent approach direction [Miller et al., 2003] .....	22
Figure 2.6: Initial robot hand postures for different grasp types [Elkval and Kragic, 2007] .....	23
Figure 2.7: Human grasp training system and robotic replication of human grasp in simulation [Elkval and Kragic, 2007] .....	24

Figure 3.1: Swiss Ranger camera [Mesa Imaging, 2011] .....	30
Figure 3.2: Data losses from specular reflectivity in red circle .....	32
Figure 3.3: Microsoft Kinect sensor [MSDN, 2012] .....	32
Figure 3.4: Microsoft Kinect mounted outside glovebox .....	34
Figure 3.5: Layers of robotic control and interaction [Kapoor, 1996].....	37
Figure 4.1: Objects in the LANL dataset. (a) broom. (b) t-fitting. (c) lathe tool. (d) large can. (e) large bowl. (f) medium can. (g) scale. (h) small bowl. (i) small can. (j) sealing tape. [O’Neil, 2013] .....	42
Figure 4.2: Gripper axes as defined relative to the Robotiq .....	46
Figure 4.3: Grasp orientations about the lip of a bowl object.....	48
Figure 4.4: Grasp orientations about the sides and top of a cylinder object.....	50
Figure 4.5: Grasp orientations about the sides of a box object.....	52
Figure 5.1 The IRAD dual arm system.....	54
Figure 5.2: The Robotiq and its three fingers [Robotiq, 2011].....	55
Figure 5.3: Variable finger movement for Fingers B and C [Robotiq, 2011] .....	56
Finger 5.4: Operational modes of the Robotiq [Robotiq, 2011] .....	57
Figure 5.5: Finger closing movement for the Robotiq [Robotiq, 2011] .....	58
Figure 5.6: Service message definition for grasp planning.....	61
Figure 5.7: ROS grasp planning node structure.....	63
Figure 5.8: RViz output for a canister grasp plan.....	64
Figure 5.9: RViz output for a top down canister grasp plan.....	65
Figure 5.10: RViz output for a bowl grasp plan .....	66
Figure 5.11: Workspace reach for cylinders .....	68

## **Chapter 1: Introduction**

The complexity involved in using a hand to successfully grasp an object is often underestimated. For humans, this grasping process is largely automated without conscious thought by our nervous system. Robotic grasping lacks this inherent advantage. Trying to impart this organic intuition to robotic hands requires understanding of both human and robotic hands to use their strengths and compensate for weaknesses.

### **1.1 HUMAN HANDS**

Human hands are unique even amongst the other primates. Much of human dexterity owes its existence to the evolution of the saddle joint of the thumb metacarpal. This joint enables humans to create opposition between fingers and the thumb and by which most of prehensile grasping is achieved. Grasp execution is very intuitive from the human perspective; when one wants to grab an object, one simply does so.

Many robotics hands have been developed according to biological inspirations. The clear advantage of such a design is their derivation from a working model that is known to be versatile and reliable - why redesign a mechanism that has seen millions of years of evolutionary tuning? In addition to purely robotic fields, human inspired hands have been explored in prosthetic fields. Early designs of prosthetic hands typically represent simple grippers comprised of one or two Degrees of Freedom (DoF) (much like early robotic grippers). Adoption and satisfaction with such devices remain meager in part due to low functionality (psychological and cosmetic effects also play a role). Human hands have a high degree of adaptability as shown in Figure 1.1 [Massa et al., 2002], and simple prosthetics lack the range of function desired by a human operator.

The move towards a more human-like hand offers more possibilities in the types of tasks that can be completed – an advantage that would provide great benefits to a robotic system where some degree of flexible automation is desirable.

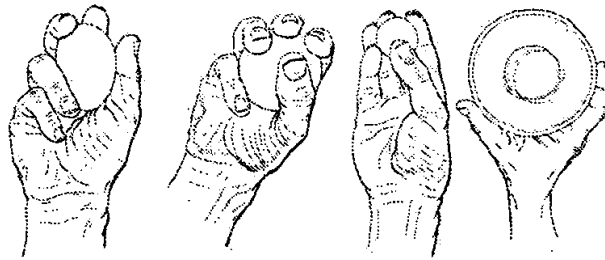


Figure 1.1: Human adaptability in grasping [Massa et al., 2002]

## 1.2 ANTHROPOMORPHIC ROBOTIC HANDS

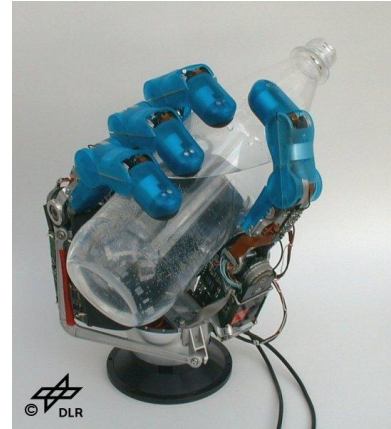
There have been numerous attempts to develop anthropomorphic grippers in academic, research, and industrial robotic communities. One currently available gripper that perhaps most accurately resembles a human hand is the Shadow Dexterous Hand in Figure 1.2a by the Shadow Robot Company. It was designed to be as similar as possible to the average hand of a human male and achieves this in its dimensions and similar degrees of freedom (the Shadow Hand has 20 versus the 21 present in a human hand [Jones, 2006]). Other hands such as the Meka's H2 Compliant Hand Figure 1.2b show a clear inspiration from a human hand without attempting to reproduce it exactly. The Meka hand contains only five degrees of freedom, but can mimic human function in its size and compliance of its fingers. The DLR Hand II in Figure 1.2c is similar in form to the Meka hand with its four fingers but each finger is four joints and three degrees of freedom.



(a) Shadow Hand



(b) The Meka hand [Meka, 2012]



(c) The DLR Hand II [DLR, 2013]

[Shadow Robot Company, 2013]

Figure 1.2: Examples of dexterous hands

### 1.3 NONANTHROPOMORPHIC ROBOTIC HANDS

Most robotic hands are not analogous to human hands. Nonanthropomorphic designs can potentially exceed human limitations on ranges of motion and be adapted to particular tasks. Such systems may lack the true versatility of a human hand, but make up for it in their reduced complexity which aids control and economic factors. Current hands that fall into this category are the BarrettHand by Barrett Technology Inc. in Figure 1.3 and the Robotiq hand shown in Figure 1.4. Both grippers are three fingered and feature joints that allow some of the fingers to rotate. For example, in the case of the BarrettHand this allows all three fingers to rotate to one side to enable a hook grasp style. There are other hands that fall into this category of nonanthropomorphic although they typically lack the dexterity and manipulability provided by the previously mentioned fingered designs. It is their dexterous limitations which preclude their use in this work. Examples of such grippers would be parallel jaw grippers and other, more novel, grippers such as the coffee ground-filled balloon gripper from Cornell University (Figure 1.5)



Figure 1.3: The BarrettHand in various stages of finger rotation [Barrett Technology Inc., 2011]



Figure 1.4: The Robotiq hand [Robotiq, 2011]



Figure 1.5: (a) Parallel jaw gripper [Robotiq, 2013], (b) Balloon and coffee ground gripper [Ju, 2010]



## 1.4 GRASP PLANNING

While a purely empirical (or human mimicking) approach to grasping will provide insight, it will not exemplify the whole picture. There must also be some analytical approach to the interaction of hand and object from first principles. The goal is not to replicate human mastery of grasping *vis a vis*, but to impart a human level of reliability and planning to a robotic hand thus increasing system autonomy. Doing so would reduce the operator's burden. If a robotic hand knows a set of valid grasps for an object and operation (a valid grasp must support the grasped object throughout the entire task), the operator merely needs to command the robot to grasp that object. Thus autonomously generating a set of valid grasps in an architecture agnostic manner elevates the entire process to a human level of functioning by reducing the robot-specific configuration information that must be dictated by the operator. Grasping has been studied extensively and many fundamental analytical issues have already been solved (e.g. quantitative grasp metrics), but grasp execution is not always translated well for the ease of human layperson operation. The literature review in Chapter 2 will largely focus on this topic as it is most related to the objectives of this effort.

### 1.4.1 Grasp Planning Suggested Sequence

The initial phase for a robot to grasp something autonomously is to identify the object to be grasped. This step can be accomplished in a number of ways in practice. Many situations exist in which the object to be grasped is known beforehand such as industrial automation. A highly specialized assembly line manufacturing robot may only need to manipulate one part which effectively trivializes the grasping problem through precomputation and repetition. In a more generalized setting, the robot must know approximately what the object is and where it is located. One possible way to achieve this is through the use of visual feedback to identify the shape and pose of the object.

Once an item has been identified, the next phase is for the robot to plan the grasp. There exist many different algorithms for analytical synthesizing grasps presented in more detail in Chapter Two. Of particular note is the planning of grasps based on the object's shape and size to balance forces applied within the object itself to achieve stability. This strategy leverages the geometric dimensions of the object against the kinematics of the robot and robotic hand. Given a system which primarily relies on visual sensors to assess the shape of objects this approach offers cohesiveness with this type of grasp planning.

The final phase for autonomous grasping is execution and validation of the grasp. At this point most of the planning is accomplished and the robot just needs to be commanded to perform the task. However, one must plan for the possibility that the robot grasp failed to accomplish its assignment as intended. The reasons that a grasp could fail can be attributed to various events such as an incompatibility with the current kinematic landscape of the robot and hand, errors in accurately determining the world model for the robotic workspace, or any other unseen error which unknowingly prevents the successful grasp. To account for such occurrences there must be a way to check that the grasping of the desired object actually happened. The methods for achieving grasp validation are further explored in the later chapters.

## **1.5 ENVIRONMENTAL CONSTRAINTS**

Grasping objects previously unknown is a challenging problem. However, in some situations this challenge can be alleviated due to environmental constraints. For this particular application the domain is the nuclear industry and specifically handling items inside gloveboxes. Such an environment poses unique dangers but allows us to make certain assumptions that reduce the scope of the grasping problem.

### 1.5.1 Nuclear Considerations

Gloveboxes are used within the nuclear domain to house dangerous radioactive materials. These materials can be dangerous due to their inherent radioactivity or secondary reasons such as reactivity (i.e. necessitating that the gloveboxes be filled with inert noble gases like argon). Radiation workers are limited in their annual exposure to radiation for health and safety. To adhere to these limits it is a federal regulation to keep doses *As Low As Reasonably Achievable*, (ALARA, *Code of Federal Regulations*, title 10, sec 20.1003). ALARA guides both operators and system designers to curtail exposures to radiation as far below annual dose limits as it practically possible when accounting for factors such as health and safety, economics, and available technology. ALARA operates on three main principles to reduce dose: minimize time in the presence of harmful radiation, maximize distance to the radiation source, and utilize shielding to attenuate the radiation. Gloveboxes primarily employ shielding to reduce dose. The interior of a typical glovebox is shown in Figure 1.6.



Figure 1.6: The interior of a glovebox [Thompson, 2002]

If Special Nuclear Material (SNM) is extremely radioactive, the material must be handled in an enclosed structure called a *hot cell*. Hot cells use both shielding and distance with master-slave manipulator systems. A radiation worker utilizing such a master-slave system for a hot cell is shown in Figure 1.7.



Figure 1.7: A hot cell and worker using telemanipulation [SCK CEN, 2013]

The master-slave manipulators operate based on transferring an operator's movements outside the hot cell to the inside space via a system of cables. These manipulator systems are almost robotic in their mechanical linkages (and some modern telemanipulator systems, such as those from Walischmiller HWM, include electronic servos furthering their robotic nature). It follows then that electrically actuated robotic systems could offer the same health and safety benefits according to ALARA if they are deployed in these radioactive environments. Transferring the burden of exposure to a robot does present challenges. The reliability of robot may be compromised by radiation damage to electrical components (single-event upsets, electronic noise, etc.) and material changes (i.e. radiation hardening and embrittlement). However, such problems are largely out of the scope of this work and are not so severe to prevent future system integration.

The strongest benefit from working in a glovebox from a grasping perspective is the rather limited set of items that may be encountered. Gloveboxes used for production have a specified set of operations they are equipped to handle. These range from analytical chemistry techniques to mechanical reduction of resources. Due to material limitations, most tools and objects found are constructed out of metals and especially stainless steel. It is therefore reasonable to assume that objects grasped inside a glovebox environment are rigid. Deformable objects can complicate grasping algorithms considerably. The hardware used for this effort is capable of grasping deformable objects, but the proposed algorithms would need to be modified to consider deformable primitives. Additionally, the tasks motivating this effort require reasonably accurate pose estimation which is not possible with a many classes of deformable objects (such a bag, for example). Thus, their elimination from the set of possible items considered dramatically improves the reliability without significantly reducing the capabilities of the robotic system in the environment of interest. Many items encountered also do not have complicated geometries. A typical operation inside a glovebox might include picking up a metal can and pouring its contents into another container. A metal can will have many different successful grasps due to its high amount of symmetry.

In summary, specialized structures have historically been used for the handling of SNM. These structures offer shielding from the radiological hazards contained inside. For especially dangerous SNM, hot cells are used to completely remove a human worker from the radiological workspace. However, human workers are allowed to interact with SNM in gloveboxes. Gloveboxes are the focus of the work contained here, and a robotic glovebox system is working inside an environment designed for humans. The glovebox can be a challenging, harsh environment, but we can leverage several of its operational

and environmental constraints to better perform robotic grasping. These challenges and benefits are summarized in Table 1.1.

<b>Challenges In Glovebox Grasping</b>	<b>Benefits from Glovebox Grasping</b>
<ul style="list-style-type: none"> <li>- Harsh environment</li> <li>- Little room for failure</li> <li>- Confined space</li> <li>- Clutter in space</li> <li>- Small but real risk of failure due to radiation</li> <li>- Potential for thermal issues due to restrictions on air cooling (flow), or inert atmosphere.</li> <li>- Difficult sensor integration for environmental monitoring.</li> </ul>	<ul style="list-style-type: none"> <li>- Rigid objects</li> <li>- Simple object shapes/Symmetry</li> <li>- Limited number of objects</li> <li>- Existing ergonomic reach limitations</li> <li>- Existing ergonomic payload limitations.</li> </ul>

Table 1.1: Challenges and benefits for grasping inside a glovebox

While grasping in a glovebox has some unique challenges (presence of SNM, co-robotic activities, etc.), many of the challenges (confined space, etc.) and benefiting assumptions (few objects of interest, rigid objects, etc.) are common in manufacturing and other application areas. Thus, the algorithms developed here – focused on glovebox activities – will likely be applicable in other application domains. It is also worth noting that most of the challenges exist even if an operator completes the task.

## 1.6 OBJECTIVES

The objective of this work is to construct a grasp planning algorithm that will be suitable for a glovebox environment that exploits the benefits derived from its particular environmental constraints while simultaneously addressing its unique challenges. To achieve this aim, items that are typically found in a glovebox and the environment itself must be well understood. After using visual information to identify objects and their

positions in the glovebox, the algorithm should return the position and orientation to which the gripper should be moved to for grasping a specified object. The grasp planner is not responsible for validating the robotic system's capability to execute the calculated grasping, but must interact sufficiently with software modules that validate the robot's capacity to execute such motions. If the grasp plan cannot be validated, the grasp planner ideally generates additional grasping options as needed.

## **1.7 STRUCTURE OF THESIS**

The remainder of this work explores the problem of grasping as applied to a glovebox environment when visual information is provided about the workspace. Chapter Two reviews the literature on robotic grasping and grasp synthesis. Chapter Three outlines the *a priori* information needed for grasping, sensing technologies, and software frameworks that will provide the data and basis for the proposed grasp planner. Chapter Four presents the proposed grasp planning algorithm. The implementation and demonstration of the grasp planner is presented in Chapter Five as well as discussions on its effectiveness. Chapter Six summarizes the efforts undertaken and outlines future work.

## **Chapter 2: Literature Review**

Robotic grasping was born along with the arrival of industrial robotics. The Unimate, released in 1961, was the first commercially available robot, and its rudimentary gripper was used to handle die castings on an assembly line [Deb and Deb, 2010]. More sophisticated grippers would come with the advent of more advanced electronics and technology [Wolf et al., 2005]. However, with more advanced grippers would come the need to more fully understand the kinematics of grasping in order to plan successful grasps.

The type of objects that a robotic hand might grasp is basically an infinite set, and so language had to be developed to be able to discuss grasping in a general sense. From this language came ways to discuss the quality of a grasp in its ability to effectively stabilize an object. This chapter describes some of the earlier work in robotic grasping, and then expands on a few particularly relevant papers to grasp planning in this research.

### **2.1 EARLY GRASPING WORK**

Up until around the 1970s, most robotic hands had simple geometries of the parallel jaw variety. Starting in the later 1970s, pioneering work was carried out by Asada and Hanafusa (1977) and Salisbury (1982). Their respective works focused on the capabilities that three (or more) fingered hands could bring to the field. An early diagram of a hand design from Hanafusa and Asada is shown in Figure 2.1.



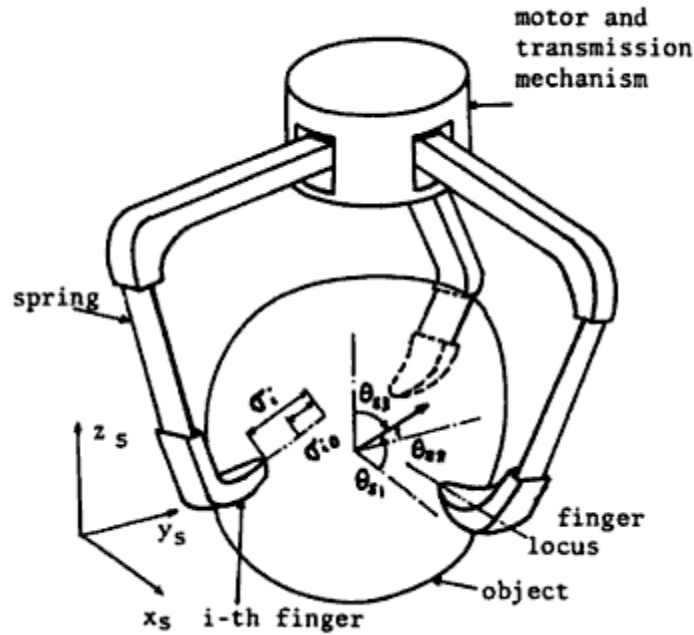


Figure 2.1: Early multifingered hand design [Hanafusa and Asada, 1977]

These early works were important not just because they helped to establish the field of grasping. These works also led the way to a better understanding of the mechanics involved in grasping and why reductions in complexity could be a valuable tool.

To understand how humans choose grasps, Cutkosky (1989) constructed a grasp taxonomy representing a systematic arrangement of possible human grasps which built off previous work by biologist Napier (1980). Starting with the two basic grasp categories described by Napier as power and precision grasps, Cutkosky observed machinists as they worked and recorded their grasp choices for various tools. The result was a catalog of grasps arranged by similarity as shown in Figure 2.2. While the taxonomy was not exhaustive, it created a baseline of grasps that helped define why one grasp might be used over another (e.g. more precision, less effort, etc.). Comparing

analytically constructed robotic grasps with those used by humans enables the analysis of the effectiveness of such analytic modeling and what grasp types are prone to slipping of the fingertips [Cutkosky and Howe, 1990]. The nature of underactuated hands lends itself to favor the power grasp style where the fingers hold an object against the palm of the hand (or base of the hand if no true palm is present).

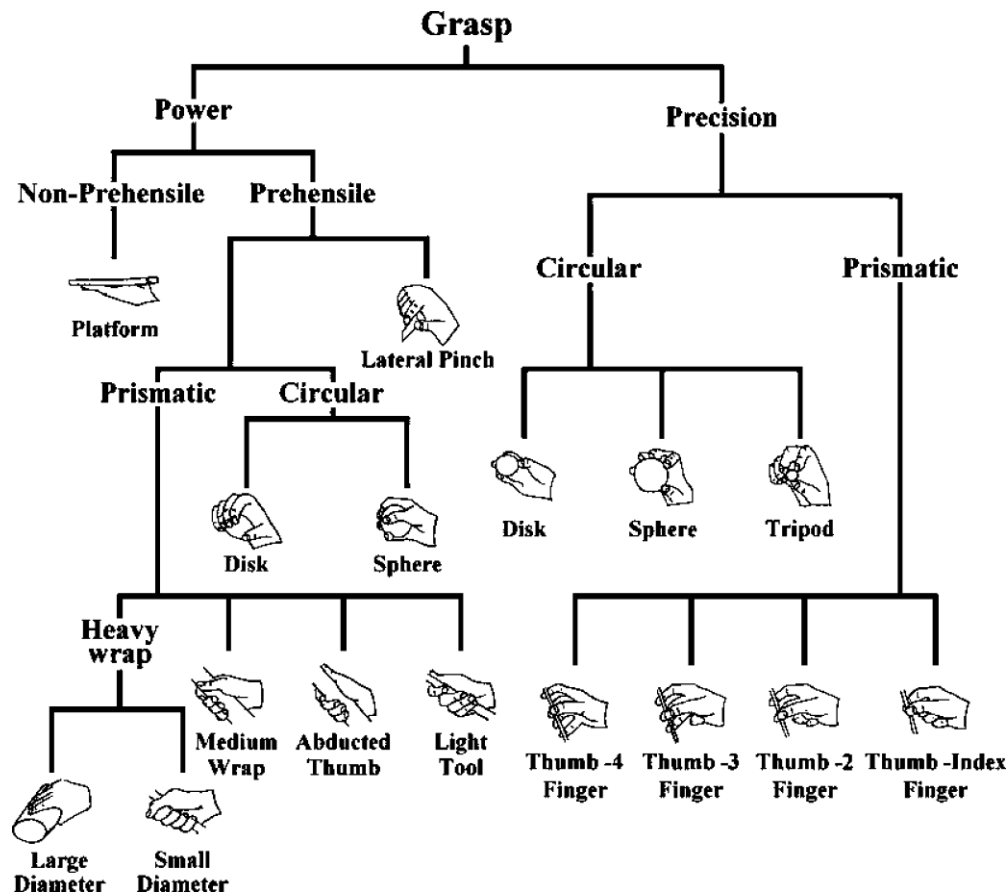


Figure 2.2: Simplified version of Cutkosky's grasping taxonomy [Murakami et al., 2009]

## 2.2 CLOSURE PROPERTIES OF GRASPS

In order for an object to be considered “grasped” it generally must be in some form of equilibrium in contact with a robotic gripper. The successful ability of a grasp to exert a static equilibrium on a grasp object is termed *force closure* or *form closure* depending on the contact mechanics involved. The contact between gripper and object is generally assumed to be a set of  $N$  points. These points can be modeled as frictionless points (force is only along common normal), frictional points (normal and tangential forces), or a soft contact (there can be a pure torsional moment exerted about a common normal) [Salisbury, 1982]. A wrench vector can be constructed for each contact point representing the three forces and moments at each into a  $6 \times 1$  vector. The unit wrenches are  ${}^i w_N$ ,  ${}^i w_T$ ,  ${}^i w_\theta$  and correspond to the forces for the normal, tangential, and moment about the normal for the contact, respectively. The corresponding magnitudes of the unit wrenches are  ${}^i c_N$ ,  ${}^i c_T$ ,  ${}^i c_\theta$ . These components can be combined into a vector of wrench magnitudes  $c$ . All unit wrenches can be combined into a wrench matrix  $W$ . When the possibility of an external wrench acting on the system,  $g$ , is included, the static equilibrium of a grasped object can be defined. A grasped object is in equilibrium if and only if a) the contact forces and moments satisfy the contact constraints for all  $i$ :

$${}^i c_N \geq 0$$

and b) the object is in equilibrium:

$$Wc + g = 0$$

These equations represent that the object has coefficients of friction sufficient enough to remain in contact with the gripper and the internal and external wrenches balance out. Building off the definition of static equilibrium, a grasp is said to be *force closed* if and only if it is in equilibrium for any arbitrary wrench  $\hat{w}$  [Nguyen, 1988]. Therefore, for any

arbitrary wrench  $\hat{w}$ , there exists an intensity vector  $\lambda$  that satisfies the contact constraint equality such that:

$$W\lambda = \hat{w}$$

*Form closure* is an extension of force closure but a grasp is said to be *form closed* if it can resist any external disturbance wrench irrespective of the magnitude of contact forces:

$$W\lambda = 0$$

Thus, form closure is a force closure with frictionless contacts [Trinkle, 1992]. The minimum number of contacts needed for form closure has been studied. As early as in 1875 it was shown by Reuleaux that at least four frictionless contacts are needed to create a form closure on planar bodies, and that some shapes are not capable of being constrained by frictionless contacts [Bicchi and Kumar, 2000]. In 1989, it was shown that most spatial objects can be form closed by seven frictionless contacts [Markenscoff and Papadimitriou, 1989]. Much work has been done in the field to calculate contact locations to achieve form closure and thus good grasp quality that is not dependent upon frictional forces [Markenscoff and Papadimitriou, 1989], [Nguyen, 1988], [Ponce and Faverjon, 1995], [Ponce et al., 1997].

The stability of grasps is a very important metric of grasp quality. While force closure is a good quality of a grasp, it does not strictly imply stability [Bicchi and Kumar, 2000]. Grasp stability can be affected by the local properties of a grasp, the force distribution, and locations of contact points and contact normals. Salisbury concluded that if the stiffness matrix (which characterizes the grasp) is positive definite then the grasp is stable [Salisbury, 1982]. However, mathematically, force closure grasps can always be made stable by adjusting the forces applied by each finger if each contact point

is treated as a virtual spring [Nguyen, 1988]. Thus, the ability of a grasp to exert certain forces on an object is a secondary measure of grasp quality.

Much of the work mentioned up to this point dealt with the assumption that all grasping contact locations could be chosen. Certainly when there is full hand control grasping strategies can include criteria-based optimization where the criteria is based on stability, for example. However, underactuated hands have seen greater use in recent years. Such hands have fewer actuators than DoF, and thus do not have the ability to impart arbitrary motions or forces at all contacts between gripper and gripped object. When a grasp is lacking the ability to command any arbitrary force the Jacobian matrix of the hand is not full rank, and the grasp is termed *kinematically defective* [Bicchi, 1994]. Most enclosing power grasps are kinematically defective [Bicchi and Kumar, 2000]. However, this “defectiveness” is actually a benefit depending on the goal of a grasp. The more a grasp is defective, the lower it is in its manipulability of the grasped object. More importantly, however, the more a grasp is defective, the lower it is in its sensitivity to positioning errors in finger placement and the more robust it is in restraining external disturbances [Bicchi and Kumar, 2000]. Therefore, underactuated hands lose the ability to have fine control over finger placement and force (i.e. low manipulability), but gain a more significant reduction in control complexity and an increase in grasp robustness.

### **2.3 SHAPE PRIMITIVES**

It is shown that a stable grasp is the optimal solution for planning a grasp, but the issue of synthesizing such a grasp can be complicated. Ideally the resolution would be to match the solution space of force closures for a particular object with the configuration space of a particular gripper, but such spaces can be dauntingly complicated and offer multiple solutions. One manner in which humans simplify the task is to form a

prehensile pose, or *pregrasp*, before engaging an object of a particular shape. This approach builds from the grasping taxonomy as created by Cutkosky. Selecting the pregrasp depends on the shape of the object to be grasped. This approach has been applied to many instances of research in the past. Rao et al. [1989] presented early work involving segmentation of three dimensional data into generalized cones which could then be formed into one of six primitive shape types and grasped based on heuristics depending on shape type. Yamanobe and Nagata [2010] successfully applied a shape primitive grasping approach to a parallel jaw style gripper. Huebner and Kragic [2008] selected gripper pregrasps by limiting the primitive representation to box shapes. Nieuwenhuisen et al. [2012] used subgraph matching to identify shape primitives based on objects identified by comparing a 3D point cloud to CAD models. Eppner and Brock [2013] match objects with successful, general grasp preshapes to exploit shape adaptability (“lets the fingers fall where they may”) of a hand for robust grasping. The next two sections explain in more depth two shape primitive approaches to grasping.

### **2.3.1 Miller, Knoop, Christensen, and Allen**

Miller et al. [2003] reduced the complexity of selecting a grasp by creating a rule based system that selects a hand pregrasp based on a simplified shaped-based model of an object. Robotic grasp preshapes depend on the geometry of hand being used: Miller et al. worked with a three fingered, 4 DoF BarrettHand. The BarrettHand is nonanthropomorphic since its fingers can rotate about its base. The paper identified four basic preshapes for the BarrettHand shown in Figure 2.3.

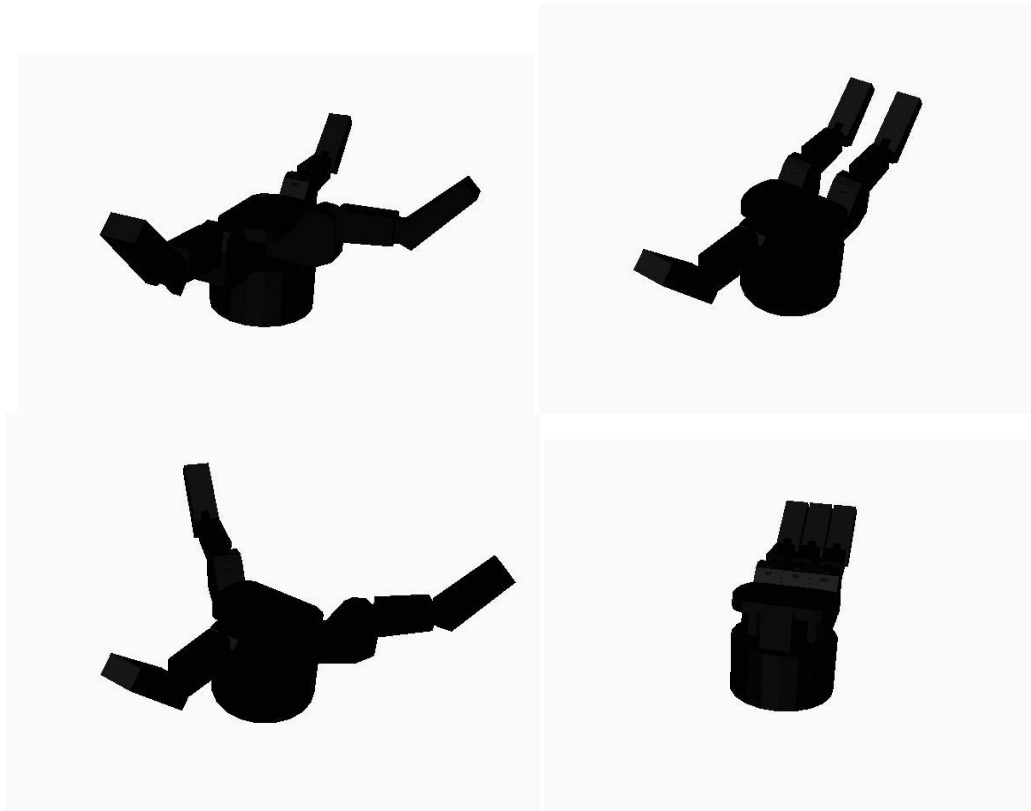


Figure 2.3: Spherical, cylindrical, precision-tip, and hook pregrasp shapes for the BarrettHand [Miller et al., 2003]

Only the first two preshapes, spherical and cylindrical, are suitable for power grasps. To pick a pregrasp shape, the method requires an approximation of an object in terms of its shape primitives. Shape primitives are basic shapes such as spheres, cylinders, cones, and boxes. An object of somewhat complicated geometry like a coffee cup is reduced to a simpler cylinder with attached box as shown in Figure 2.4.

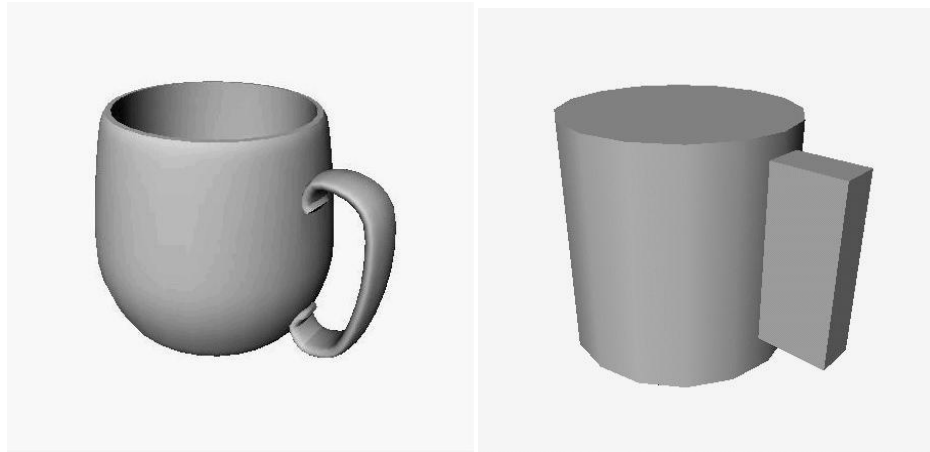


Figure 2.4: A model of a coffee mug and its associated model from shape primitives  
[Miller et al., 2003]

Miller et al. created a heuristic set of rules to deal with each shape primitive type which effectively limited the large number of possible grasps for such shapes while still providing successful grasps. Their grasping rules are detailed in Table 2.1.



<b>Primitive Type</b>	<b>Grasping Strategy</b>
Box	Cylinder pregrasp shape such that the two fingers and thumb will contact opposite faces. The palm should be parallel to a face that connects the two opposing faces, and the thumb direction should be perpendicular to the face it will contact.
Sphere	Spherical pregrasp shape. The palm approach vector should pass through the center of the sphere.
Cylinder	<p>a) Side Grasp: Cylindrical pregrasp. The grasp approach should be perpendicular to the side surface, and the thumb should either be perpendicular to the central axis of the cylinder, in order to wrap around it, or in the plane containing both the approach direction and the central axis, in order to pinch it at both ends.</p> <p>b) End Grasp: Spherical Pregrasp. The palm should be parallel to the end face and aligned with the central axis.</p>
Cone	Can be grasped in same manner as a cylinder. For cones with large radius and small height, the side grasps will be similar to a grasp from the top – in this case there is a set of grasps around the bottom rim of cone, where the palm approach vector is aligned with the bisector of the angle between the bottom face and side face.

Table 2.1: Shape primitives and associated grasping strategies, adapted from [Miller et al., 2003]

The grasping strategies are represented visually in Figure 2.5.

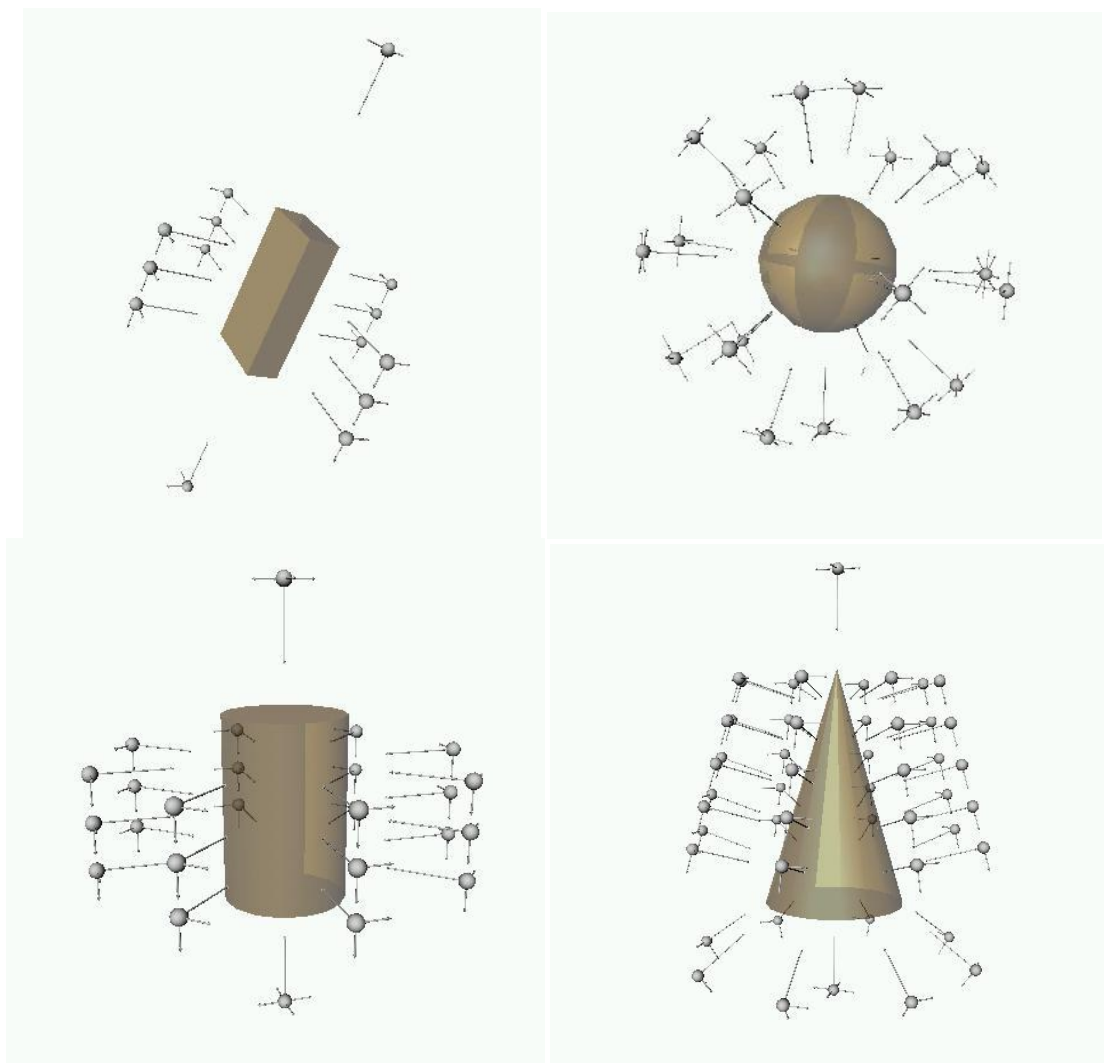


Figure 2.5: Examples for grasp generation on the four shape primitive types. Balls represent starting positions for the center of the palm, and long arrows represent approach direction [Miller et al., 2003]

Miller et al. used their own grasping software, GraspIt! to evaluate the generated pregrasps, and were able to successfully plan valid grasps. Some drawbacks to their method include the need to know all objects beforehand (not always a feasible assumption) and the long processing time needed to determine the best grasp (from tens of seconds to minutes) [Miller et al., 2003].

### 2.3.2 Elkvall and Kragic

Elkvall and Kragic [2007] expanded on the usage of shape primitives from Miller et al. Elkvall and Kragic used the same BarrettHand as Miller et al. but also introduced the use of the anthropomorphic Robonaut hand. The researchers strove to use pregrasp shapes that closely emulated human pregrasp shapes in an attempt to make the grasping more “natural.” Their robotic pregrasps and related human poses are shown in Figure 2.6.

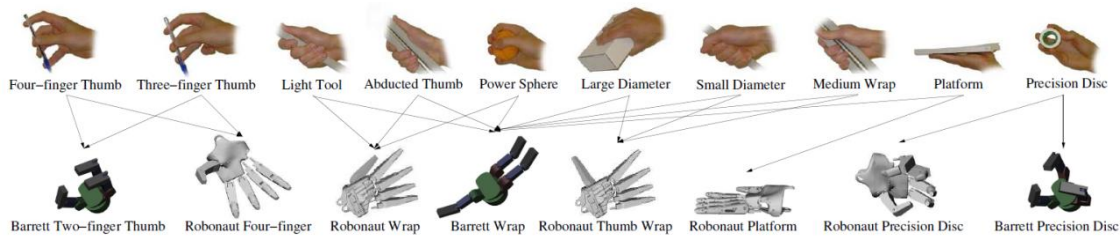


Figure 2.6: Initial robot hand postures for different grasp types [Elkvall and Kragic, 2007]

Elkvall and Kragic used a glove with magnetic trackers worn by a human to initially pick up objects from a table as shown in Figure 2.7. The robotic system then mapped that type of grasp used to that object and attempt to replicate the grasp robotically. Using this approach the robotic system was trained on what types of grasps work best for what type of primitive. Using a vision recognition system that had an object recognition rate of 100% for five objects, the system was able to provide suitable grasps 96% of the time for ten grasp types. Notably, the authors also introduced errors in pose estimation to assess how well the system could react in such cases when the vision recognition system is not completely accurate. The authors found that power grasps were less sensitive to pose errors than precision grasps (as to be expected with kinematically defective grasps), and

that some grasp types were not as sensitive to pose errors when being executed around an axis of symmetry for some objects.



Figure 2.7: Human grasp training system and robotic replication of human grasp in simulation [Elkvall and Kragic, 2007]

## 2.4 SUMMARY

Early grasping focused on the particular control of individual contact points for grasping. Having the ability for such fine control gives a robotic operator a lot of control over the success of a grasp, but such control can be overwhelming. Kinematically defective grasps of the type that underactuated hands favor tend to be more robust than fingertip precision grips. These grasps with large object-hand contact areas do not have the ability to finely manipulate objects, but can be favored when item integrity and handling safety are very important (i.e. objects in a nuclear environment).

The examples of shape primitive grasping systems (given in more detail above) offer advantages when the types of items that will be encountered are well understood. As mentioned before, due to the tightly controlled regulation of gloveboxes, it is unlikely that an object would be encountered in such an environment that is novel. This makes a

shape primitive based approach promising for a nuclear robotics system over designing an inextensible strategy which precalculates gripper contact points. For an underactuated gripper such as the Robotiq, shape primitive based grasping makes sense because pregrasp shapes are effectively built into the design of the gripper. The Robotiq has multiple operation modes that alter the configuration of the fingers to different shapes, and will be explored as a mechanism for this grasping strategy in a later chapter.

## **Chapter 3: *A Priori* Data, Sensing data, and Robotic Operational Software.**

Grasping in a glovebox takes place within a larger scheme of a robotic system. That is to say that it is merely one part of the puzzle that is the successful implementation of a robotic solution for a nuclear environment. Multiple subsystems must be present to properly replicate tasks that are currently performed by human radiation workers. Grasping is one of these subsystems if dexterous manipulation is to be required. Other potential subsystems include computer vision and recognition, motion planning, force control, and human-machine interaction. These subsystems are often interdependent.

This chapter reviews those subsystems that the proposed grasp planner must interact with in order to 1) collect the information necessary to determine a valid grasp, 2) validate the grasp, and 3) execute the proposed grasp. Since the capabilities and interfaces for these subsystems will impact the requirements and thus the development of the grasp planning algorithm, the instantiations of these subsystems used for this research are reviewed in detail. Thus this chapter will present each category (*a priori* data, sensor data, and robotic operational software frameworks) first generally and then specific to the environment and test bed currently used at the University of Texas at Austin and proposed for use at LANL.

### **3.1 *A PRIORI* INFORMATION**

Grasping requires geometric and other information for the object to be grasped and where that object is located relative to a known frame of reference. There are different ways this knowledge can be imparted to a grasp planner. Having a graph based world model, for example, can supply a grasp planning algorithm with all the relevant

information it may need [O’Neil, 2010]. However, the problem then becomes what technology is suitable to populate the information needed in the world model.

One solution is to have the operator supply the necessary information regarding object position and type. Yet, this practice creates a new burden on the operator. For applications where it is desired to newly incorporate robotics, the robotic solution should result in less overall work for the system operator than the previous solution. This is to help ensure successful adoption of the technology. To ease into a robotic solution, it can be advantageous for a smoother transition to increment the amount of autonomy provided by a system. This has been termed “transitional levels of autonomy” and is shown in Table 3.1 [Brabec, 2011].

Level	Tele-operation → Autonomy...
1	Reduce or eliminate operator's need to manage the robot's internal configuration.
2	Reduce or eliminate the operator's responsibility for avoiding undesired contact with the environment.
3	Reduce or eliminate the operator's responsibility for moving the robot to locations of interest.
4	Reduce or eliminate the operator's responsibility for selecting a proper grasping configuration for retrieving selected objects.
5	Allow the operator to quickly direct the system to complete tasks that involve subtasks completed as directed in levels 3 & 4 (such as pick and place).
6	Reduce or eliminate the operator's responsibility to avoid threshold forces for contact tasks such as opening a door or lifting items exceeding the system's payload.
7	Reduce or eliminate the levels of detail that are necessary for the operator to communicate a task (or subtasks) to the robotic system.
8	Integrate capability to complete tasks that require high levels of precision and/or the control of a specific force profile.
9	Reduce or eliminate the need for the operator to be in the loop for tasks that respond to non-operator, independent external events (i.e. timer on oven, low battery notification, etc.).
10	Based on prior tasks completed, the system anticipates future tasks to be completed based on historical use.

Table 3.1: Transitional levels of autonomy

The lowest level in this scheme represents a system with no autonomy, whereas the highest level represents full autonomy. Due to the level of oversight given to materially sensitive tasks within the Department of Energy, it is unlikely that those tasks will become fully automated in the near future. In such a domain, the structure of the transitional levels of autonomy and their incremental improvements offer a roadmap for the incremental incorporation of robotics. It is for such reasons that it is undesirable to force an operator to take full responsibility to instruct a robotic system what and the location of an object to be grasped. To reduce the burden on the operator, it is beneficial for the system to be able to automatically populate this object data.



Planning an autonomous grasp relies heavily on information provided by a sensing subsystem to update the world model of where and what objects to grasp as well as identify environmental complications such as obstacles, occlusion, or other factors that may impact the grasp. In this effort we will focus on grasping items without such complications. This assumption is acceptable given the tasks in a glovebox we are currently focused on. However, future efforts will need to address these complications.

### **3.2 SENSING TECHNOLOGIES**

Different solutions are possible to provide data to a grasp planning algorithm. Each has its own advantages and disadvantages. What technology to select can be shown to be largely dependent upon the application area for the robotic system.

#### **3.2.1 Tactile Feedback**

Many commercially available robotic hands are able to be fitted with tactile sensors. [Syntouch LLC, 2013], [Barrett, 2011]. Tactile sensors often are comprised of some type of transducer that is sensitive to touch, pressure, or force. Sometimes they are combined with other sensors to replicate the human tactile experience. In the case of the Syntouch biomimetic tactile sensor kits for robotic grippers, an electrode array provides force feedback, fluid and pressure transducers provide vibration feedback, and thermistors provide temperature feedback [Syntouch LLC, 2013]. Tactile feedback can be used by a robot to explore its environment. Some tactile sensors can provide textural information that could help distinguish objects from one another. [Syntouch LLC, 2013], [Howe and Cutkosky, 1993]. The issue with tactile sensors is the need for physical interaction for identification. Such interaction could prove disastrous inside a glovebox if an attempt for identification resulted in the accidental tip over of a canister of sensitive material. Additionally, most tactile sensors utilize materials or electronics that are not

ideal for glovebox environments. While tactile feedback can be useful, it is for the previously mentioned reasons that it cannot be integrated alone; an accompanying sensing technology must either augment or replace tactile feedback.

### **3.2.2 Computer Vision**

Computer vision is another sensing technology that can build the world model but does not require physical interaction like tactile sensing. Optical sensors can survey a workspace and use various algorithms to both separate an object from the surrounding environment and identify what that object may be. Using computer vision to partition a scene into multiple useful segments (such as isolating objects) is termed image segmentation. Segmentation is a field of computer science in its own right, and has seen much use in the field of grasping to identify objects [Saxena et al., 2008], [Richtsfeld and Vincze, 2012], [Rao et al., 2010], [Varadarajan, 2011]. While the algorithms used in computer vision and recognition may have some generalities across various types of cameras, not all camera types are suitable for nuclear applications. One type of vision sensor is an infrared camera such as the Swiss Ranger made by Mesa Imaging AG shown in Figure 3.1 [Mesa Imaging, 2011].



Figure 3.1: Swiss Ranger camera [Mesa Imaging, 2011]

This camera is a 3D time of flight camera, meaning that the camera emits infrared light which travels to objects in a scene, reflects off those objects, and travels back to the camera. The time of arrival of the reflected light is measured independently by each of tens of thousands of pixels on the camera sensor. For many applications, such a camera could be sufficient for providing data to the world model. For nuclear applications inside a glovebox, it may not be readily apparent why such a camera would not be ideal. The interior of a glovebox is often constructed from stainless steel. Stainless steel is typically chemically resistant and able to tolerate radiation fairly well (stainless steel is used in the construction of reactor pressure vessels for commercial nuclear power reactors, and, after radioactive exposure during normal operation, will experience hardening and embrittlement over its operational lifetime). The beneficial properties of stainless steel also cause many objects likely to be found within gloveboxes to also be constructed from stainless steel. Having items that are made of the same material complicates identification for an infrared camera, particularly when the material in question exhibits a high amount of specular reflectivity. Sections of Figure 3.2 that are solid white represent regions of specular reflectivity, and are regions where no information is being recorded. It is therefore more advantageous to use a camera without such a dramatic loss of object data.



Figure 3.2: Data losses from specular reflectivity in red circle

The Kinect camera from Microsoft is a sensor that operates on the emission of infrared light similar in concept to the SwissRanger camera. The Kinect (Figure 3.3) emits a grid-like pattern of infrared dots over the scene within the camera's field of view.



Figure 3.3: Microsoft Kinect sensor [MSDN, 2012]

An infrared sensor reads back the reflected beams and converts the data to depth information (distance between the sensor and point in field of view). Coupled with the depth sensor is a RGB camera providing color imagery of the scene at a 1280x960

resolution. Additional sensors include an accelerometer and microphone array [MSDN, 2012]. While the light sensors are of the most use for populating world information, the secondary set of sensors belies the Kinect's origins as a video game peripheral. The Kinect was developed by Microsoft to be an interactive addon for their Xbox game system; players step into the field of view of the camera and the camera translates their real movements into virtual ones. The initial offering of the camera as entertainment device instead of serious research tool meant that the price point had to be within reach of Microsoft's target consumers. The relatively inexpensive price of the Kinect potentially contributed to the boom in its use in research; a Kinect can be obtained for the low price of roughly \$150 instead of several thousands of dollars for a sensor like the SwissRanger. In glovebox testing, the Kinect sensor has notably lacked the issues with specular reflectivity seen with other cameras. Furthermore, the technology used by the Kinect enables the data to be collected from outside a glovebox through the leaded glass windows (Figure 3.4), whereas the SwissRanger camera had to be mounted inside a glovebox to collect data.



Figure 3.4: Microsoft Kinect mounted outside glovebox

This camera placement is important because, due to strict Department of Energy regulations and procedures, it is expected to be far easier to mount a device outside an active, working glovebox than it would be to incorporate a novel object into the working area inside a glovebox. It is for all such reasons that the Kinect is an ideal computer vision device to collect data about a glovebox environment and objects within it that could be grasped. Additionally, a successor to the original Kinect has been announced for the upcoming next generation of video gaming systems expected around the end of 2013. The new Kinect is modified to use time of flight with emitted infrared light just like a SwissRanger [Coldewey, 2013]. It remains to be discovered if the new Kinect will maintain the advantages of the original version.

### **3.2.3 Force Feedback**

Similar to tactile feedback, force feedback cannot offer a complete object identification scheme. Tactile feedback could be considered a subset of force feedback in that it relies upon physical interaction to generate data, but the location and type of sensors differ slightly. Force information can be gleaned from a robot with a current-to-torque mapping for the robotics joints or with a force-torque sensor mounted on the robot. It is technically feasible that such systems can be used to update the world model through exploratory motions [Schroeder, 2013]. Like tactile feedback, such a solution would not be ideal for a sensitive glovebox environment. Force feedback can successfully be used to monitor for collisions between robot and environment when performing tasks, but such an application is outside the scope of this work. Force feedback that can be useful, however, is the inclusion of force data to verify grasp success as outlined in Chapter 1.

A sensor that can be utilized for force data is a six axis force-torque sensor such as the ATI Gamma series by ATI Industrial Automation. Such a device is typically mounted on the robot at the wrist just before the gripper. This sensor detects the forces and torques along and about three principal axes relative to the device.

## **3.3 OPERATIONAL SOFTWARE LIBRARIES**

As can be seen with the presence of numerous technological subsystems, a developing robotic platform can easily devolve into a complicated tangle of integration. To facilitate the union of all subsystems, the Nuclear Robotics Group has explored a couple of frameworks to act as a technical umbrella. In addition to the incorporation of sensing technologies, the actual control of the robot is also needed. While there are other operational software libraries in existence, the following libraries are explored because they offer the best combination of performance (can they provide the needed functions)

and practicality (will they or do they already work with hardware from the Nuclear Robotics Group).

### **3.3.1 OSCAR**

OSCAR was a project developed in the mid-1990s at the University of Texas to integrate various robotic control functions. The development of OSCAR grew out of a need to provide a generalized architecture that could be used for advanced robotics (e.g. redundant robot arms). OSCAR stands for *Operational Software Components for Advanced Robotics*. It was built to address the needs of the middle layer of robotic control software in Figure 3.5.



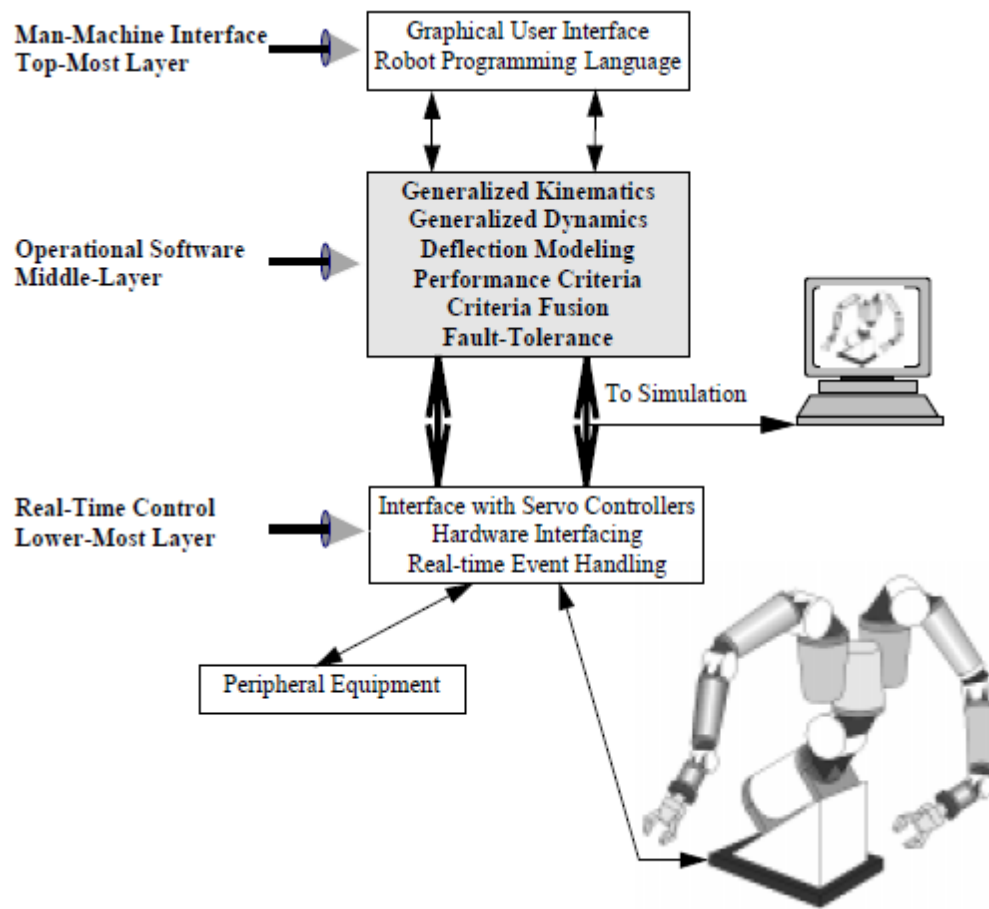


Figure 3.5: Layers of robotic control and interaction [Kapoor, 1996]

By creating a generalized software architecture that could be applied to many different robots, OSCAR can dramatically simplify the control of a robotic system by negating the need to develop custom kinematic solvers for every robot. OSCAR was shown to give dramatic performance improvements: a real-time implementation of a direct-search inverse kinematics technique showed an improvement of 170x performance for a seven DoF arm, the program development time was reduced by about 200% as compared to other robotic software in use at the University of Texas at the time, and it provided an overall 30% improvement in “ease of use” [Kapoor, 1996]. The benefits of such an

operational library are clear, and enable the robotic software developer to take advantage of the modularity and code reusability supported by OSCAR's software design methodology. Since its initial development, OSCAR has gone through several development iterations by private company, Agile Planet, and exists as the core of the rebranded library Kinematix [Agile Planet, Inc., 2013]

### **3.3.2 ROS**

ROS, or Robot Operating System, provides services one might expect in a typical operating system such as "hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management" as well as "tools and libraries for obtaining, building, writing, and running code across multiple computers" [ROS.org, 2013]. ROS was developed as a framework to address issues in the development of large-scale service robots from Stanford University and Willow Garage, but its application is more expansive than just the service domain. The original goals and characteristics for ROS were defined as follows: peer-to-peer, tools-based, multi-lingual, thin, and free and open-source. A ROS project contains multiple processes that can be run on different hosts connected in a peer-to-peer fashion. Code for ROS can be written in a number of languages such as C++, Python, and LISP. ROS encourages a high degree of modularity and follows a microkernel design (e.g. uses a number of small tools to build and run components instead of a monolithic kernel). A typical ROS system will be comprised of many nodes, which are essentially individual processes that perform some sort of computation. Communications between nodes are termed messages and must be predefined data structures. To send messages in ROS there are two options: topics and services. A node may publish its messages to a topic defined with a string name. Other nodes interested in that topic may subscribe and receive any

information published to that topic. In the topic method, publishers and subscribers are not aware of the others existence. The service method is more akin to a traditional return value from a function in computer programming. Each service will consist of a defined request and response type [Quigley et al., 2009]. ROS, at its core, is a messaging platform.

An extension of ROS is ROS-Industrial. ROS-Industrial, or ROS-I, is a collaborative project initiated by Southwest Research Institute to extend the capabilities of ROS to support industrial robotic applications. ROS-I seeks to bridge the gap between academic and industrial robotics by enabling greater support for the advanced capabilities researched in academia (e.g. machine perception and path planning instead of hard coded and predefined movement paths that cannot respond to dynamic environments). Many industrial robots and their controllers run on proprietary platforms and languages. ROS-I serves as an intermediary between the proprietary messages and other ROS messages by creating software wrapper nodes. It is hoped that future industrial robotic systems will be designed to be compatible with ROS to avoid the need for software wrappers [Edwards and Lewis, 2012].

An associated library is the MoveIt! software package available for ROS. MoveIt! is a universal framework for robotic motion planning. It provides the ability to perform collision-free motion planning, execution, and monitoring for any robot. Robot universality is implemented by a uniform XML specification structure called Unified Robot Description Format (URDF) which defines the kinematic layout of a robot [Chitta et al., 2013]. Many other tools are available in addition to MoveIt!. A key tool utilized in this work is RViz which is a 3D visualization tool (RViz for “ros-visualization”). RViz is able to interface with MoveIt! to display a robot and its motions in simulation.

RViz is also able to display primitive object types called markers which can be useful to represent simple objects in a workspace.

### **3.4 SUMMARY**

This chapter described the basic information needed before a grasp can be successfully planned. In an effort to simplify the control of a robotic system, it is desired to increase the system autonomy to a point at which the system operator is not tasked with controlling every minute action directly. It is less burdensome to use various technologies to identify what and where objects are located in a workspace. Force and tactile sensors can be used to explore the working environment, but are not ideal when minimal extraneous physical interaction is desired between system and objects. For radiological work, this interaction minimization serves to reduce the likelihood of accidental spillage of materials present in a glovebox, but can also protect a robotic system from radiation by increasing distance to a radiological source. The ideal solution is computer vision to populate the world model because it does not have to directly interact to observe, and it is also a technology which sees common use in the field of grasping. To incorporate control of sensing technologies and the robotic parts themselves, various operational software libraries exist which facilitate system development and control. OSCAR and ROS are such libraries which offer support for the high level control of robots and their peripheral devices. The modularity supported by these operational software libraries enables development work to focus on a particular subsystem such as grasping.

## Chapter 4: Proposed Algorithm Solution

This chapter presents the experimental setup and the proposed grasping algorithm. It outlines the hardware and software architectures, and the proposed grasping algorithm.

### 4.1 GRASP PLANNING ALGORITHM

The grasp planning algorithm represents the calculations contained within the *grasp planner server* node. In its simplest form, the proposed algorithm takes in an identified object and its pose and returns a strategy for grasping such an object. After calculating the grasping strategy for that object, the calculations are checked against the current state and limitations of the robot to execute a movement to the grasping locations. If the calculated grasp for the object is not feasible for the robot to perform, alternate grasps are generated and checked again. ROS and many other efforts in the literature and the research community provide the software necessary to execute (or determine the validity of) a robotic move. Thus, this effort will focus on necessary determination of determining valid grasping configurations.

#### 4.1.1 Key Assumptions

Given the application area of a glovebox, there are limitations on the types of object that will be present. Objects are generally rigid and metallic. A relevant dataset for glovebox work was created to train the vision recognition system. The LANL dataset was created by researchers at the NRG to represent pose-annotated objects typically be found in a glovebox environment. Many of the objects are constructed from stainless steel and offer little textural uniqueness which necessitates a system that can differentiate based on shape. The LANL dataset is shown in Figure 4.1. The vision system operates on the assumption that the objects are situated vertically on a planar surface.



Figure 4.1: Objects in the LANL dataset. (a) broom. (b) t-fitting. (c) lathe tool. (d) large can. (e) large bowl. (f) medium can. (g) scale. (h) small bowl. (i) small can. (j) sealing tape. [O’Neil, 2013]

There is also the assumption that the camera for the computer vision system is mounted somewhat near the base of the robot. This is the case when the Kinect is mounted outside a glovebox window near the gloveports the robot is inserted though. It would not be a great challenge to assume a different installation point for the camera, but it could change how the system must address the pose for an identified object. For example, if the closest identified point is on the opposite side of the object from the robot, the algorithm would try to grasp from that farthest side first which may have potential collisions or other unsuitabilities. However, with kinematic error checking this issue may be avoidable as the system would try to find alternative grasps that are reachable in that situation.

#### 4.1.2 Object Shape Categories

Upon closer examination of the LANL dataset, it becomes apparent that the set can be characterized by an even smaller set of primitives similar to those suggested by

Miller and other researchers as discussed in Chapter 2. The large (d) and small (f) cans are the same shape but scaled differently. The cylindrical shape of the cans is also apparent in other objects such as the sealing tape (j) and handle of the broom (a). The relative similarity between object shapes lends itself to a heuristic grasping approach based on shape primitives. Objects can be classified into one of three main categories based on the LANL dataset: box, cylinder, and bowl. The cans fit into the cylindrical category whereas the bowl is in its namesake category. We can be assured that additional categories may be necessary as additional items are considered. However, it is clear that many items will fit in these categories and the addition of new categories will not be a common occurrence. Thus, strategic use of heuristics based on these categories does not overly restrict the future extensibility of the proposed algorithm.

Because of the basis on shape primitives, not every item will completely fit a category. Some of the more exotic items, while they may have similarities to other categories, can be treated as special cases. For example, the lathe tool is one such item. Other items may be placed in different categories depending on how they are situated or how the proposed use dictated a particular grasp. For example, the roll of tape shares the basic cylindrical shape while having a lip much like a bowl; the tape can be treated as either depending on what task needs to be carried out. The breakdown of objects from the LANL dataset into shape primitive categories is shown in Table 4.1.

Shape Primitive	LANL Dataset Objects
Cylinder	Broom(handle), Large can, Medium Can, Small Can, Sealing tape
Bowl	Large bowl, Small Bowl, Sealing tape
Box	Scale (top section)
Unique	Lathe, T-fitting, Scale

Table 4.1: Shape primitive breakdown for LANL dataset items

#### 4.1.3 Kinematic Construction of Grasp

The basic construction of a grasp for any type of object is to get the gripper in a position so that when the fingers close the object will come into a stable configuration in contact with the gripper. There are two main poses that must be calculated for this to happen. The end effector pose is the gripper position as previously described; the pose in which the closing of the gripper will result in a successful grasp. The approach pose is the preliminary pose which must occur before the end effector pose is executed. The approach pose offers an initial point and orientation in space for the robot to move towards. The movement from one pose to another is linear (although this could be changed should tight space requirements dictate a more complex strategy), and the poses are structured to be a predetermined distance apart to allow room for the opening of fingers which is based on the dimensions of the gripper and its fingers. Once at the approach pose, the gripper may open all its fingers if closed. The shape of the gripper at open approach pose and end effector pose are equivalent to the idea of a pregrasp position in the literature. The approach pose is needed to prevent collision between object and gripper which may be possible at the closer proximity end effector pose. This work



operates under the assumption that this space is sufficient for the opening, closing, and reconfiguration of operational mode without collision. Given that a glovebox can be a confined space, the assumption that all finger motions occur without issue may not always hold merit. This can be addressed in one of two ways in a deployable system: utilizing the motor currents of the gripper fingers to assess if something has unexpectedly come into contact with the gripper during a finger movement and ceasing movement or by modeling the gripper as its whole physical workspace while monitoring collision detection in simulation. For visualization purposes the Robotiq gripper is shown in this chapter, but the algorithm is not limited to only this hand. The position of the gripper with respect to the robot is defined as the tool frame and its orientation in terms of the gripper axes is shown in Figure 4.2, where the z-axis comes up out of the palm. The coordinates used to define object and pose positions are calculated as if the base frame of the robot is the same as the base frame of the world - that is to say that the base of the robot corresponds with the world origin at (0,0,0). The approach frame, approach path, and grasping frame are determined based on the object's categorical primitive selected as outlined above.

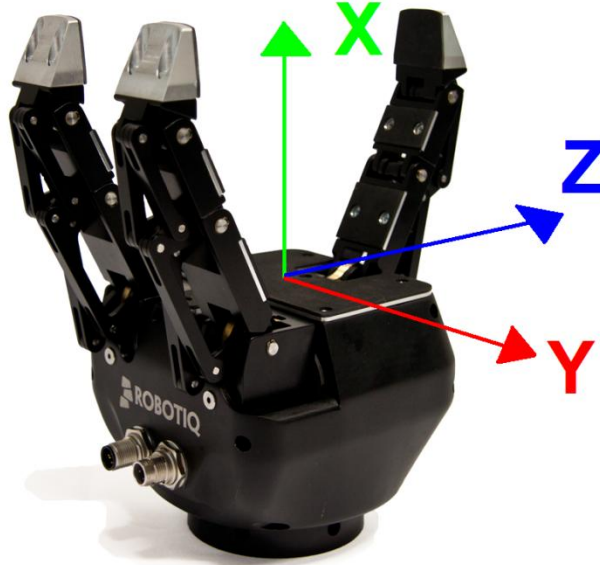


Figure 4.2: Gripper axes as defined relative to the Robotiq

For the grasping algorithm, the end effector pose is actually generated first before the approach pose. The basic process involves generating a vector from the base of the robot to the surface of the identified object:

$$\overrightarrow{v_{obj}} = P_{obj} - P_{robot}$$

Where  $P_{obj}$  is the point in space of the object and  $P_{robot}$  is the point in space of the robot base (defined as (0,0,0)). Since the point on the object identified by the computer vision is the closest point to the vision, it is not necessarily the ideal point at which to grasp. The point on the object can be translated along the object surface to a more suitable location (which is highly object dependent):

$$\overrightarrow{v_{obj\_new}} = \overrightarrow{v_{obj}} + \mathbb{R}_{obj} * \overrightarrow{v_{offsets}}$$

Where  $v_{obj\_new}$  is the new vector pointing to the new starting point on the object surface,  $\mathbb{R}_{obj}$  is the rotation matrix representing the orientation of the object, and  $v_{offsets}$  contains the object-specific offsets by which to translate to reach a suitable starting point. For

example, this will be used to create a vector going from the base of the robot to the *lip* of a bowl object instead of the side of the bowl. The idea is to get a starting point at which the process of constructing the axes of a grasp can commence – the closest side to the robot is chosen as a convenience for calculations. To construct an initial grasp, the vector to the object,  $v_{obj\_new}$ , is normalized and rotated to correspond with a known pregrasp axis via scalar dot product projection. Some space may be inserted to allow clearance for the opening and closing of fingers by translating a small amount along a pregrasp axis in the direction away from the object. Generally, since the x-axis and z-axis of the hand are determined by the grasping strategy for an object type, they are known relative to the base frame and simply need to be set around the starting point of the object (e.g. for grasping a bowl lip, the hand's x-axis would need to point downwards in the world frame, above and towards the lip). The y-axis can be constructed from the cross product of the z-axis and x-axis:

$$\vec{Y} = \vec{Z} \times \vec{X}$$

Selected code from the grasp planning algorithm can be seen in the Appendix to better clarify the calculations described in this chapter.

#### **4.1.3.1 Bowl**

When the computer vision system returns the point located on the side of the bowl, the grasp planning algorithm uses this information to calculate a vector from the base frame of the robot to the identified point. The reasoning behind this starting point for this primitive and others primitives is to provide a convenient initial point at which to begin grasp calculations. It is a point that is less likely to have collisions between robot and object than a point on the side farther away from the robot. Bowl shaped objects can be difficult for grasping depending on their orientation. The computer vision system is

trained on bowls that are upright with the opening pointed upwards, as bowls in an upside down position with the opening downwards would prove extremely difficult to grasp without the aid of an external device to reorient the object due to the mechanics and dimensions of typical robotic hands. The ideal grasping location for an upright bowl is a pinch-style precision grasp centered on the lip of the bowl. To grasp the bowl the grasping algorithm translates the calculated vector up the side of the bowl to the lip, and then applies a rotation matrix to reorient the vector to be in line with the surface normal of the top of the lip. This vector is normalized and set as the x-axis of the end effector pose. The remaining axes are constructed to be orthogonal to the lip surface normal and enable the gripper fingers to pinch against each other with the bowl lip in between. The approach pose is generated by backing the pose away from the object along the x-axis direction. The operational mode is set to pinch. Possible poses representing the x-axes in red are shown in Figure 4.3 (the other axes are represented in yellow for one case).

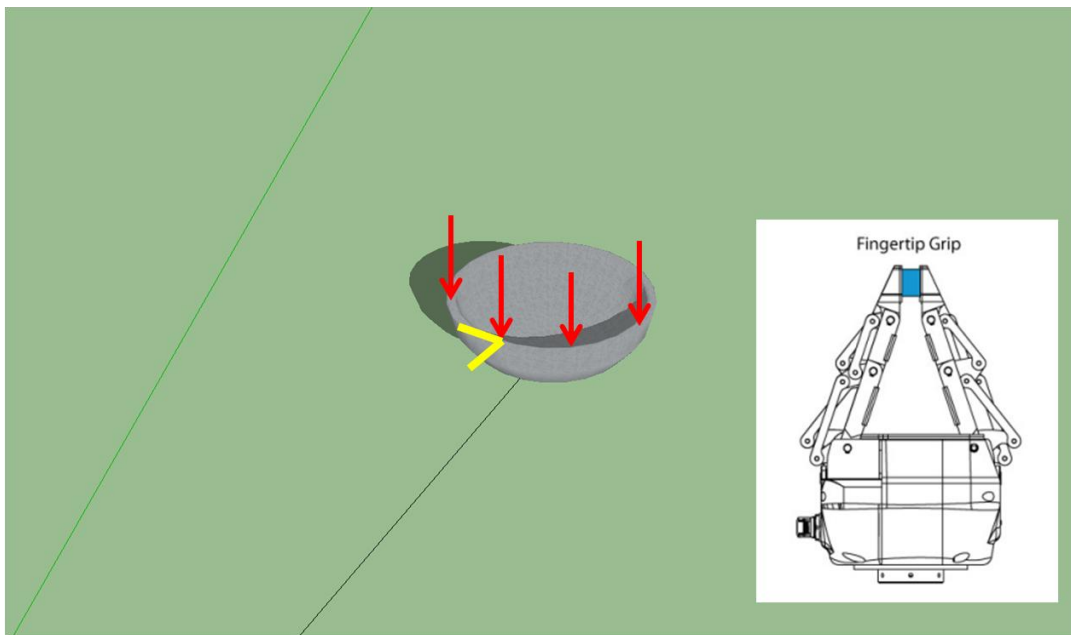


Figure 4.3: Grasp orientations about the lip of a bowl object

After the pose calculations, the inverse kinematics for each are checked to make sure each is feasible. These checks are based on just the two grasping poses and not the entire trajectory the robot must take to move into position. If, for some reason, the inverse kinematic check should fail for either, new poses will be generated. For the case of the bowl, the poses are rotated about the center of the bowl along the lip while taking the bowl's radius into account. The rotations continue about the lip surface until a suitable set of poses is found. If no set of poses can be calculated for a successful grasp (e.g. the bowl is out of the reachable workspace of the robot), an error is returned to the main ROS node that requested grasp planning.

#### ***4.1.3.2 Cylinder***

Similar to the bowl case, a vector is cast from the base of the robot to the identified (via computer vision) point on the cylinder and rotated to be parallel with the horizontal base plane of the world. This vector serves as the initial x-axis for the end effector pose. The remaining axes are constructed to be parallel with the vertical plane of the robot base frame and orthogonal to the surface normal at the x-axis and object intersection. The gripper fingers are positioned to close around the vertical sides of the cylinder. The approach pose is formed by backing out the pose along the x-axis of the gripper frame towards the base of the robot. The operational mode is set to basic. Example poses representing the x-axes in red are shown in Figure 4.4.

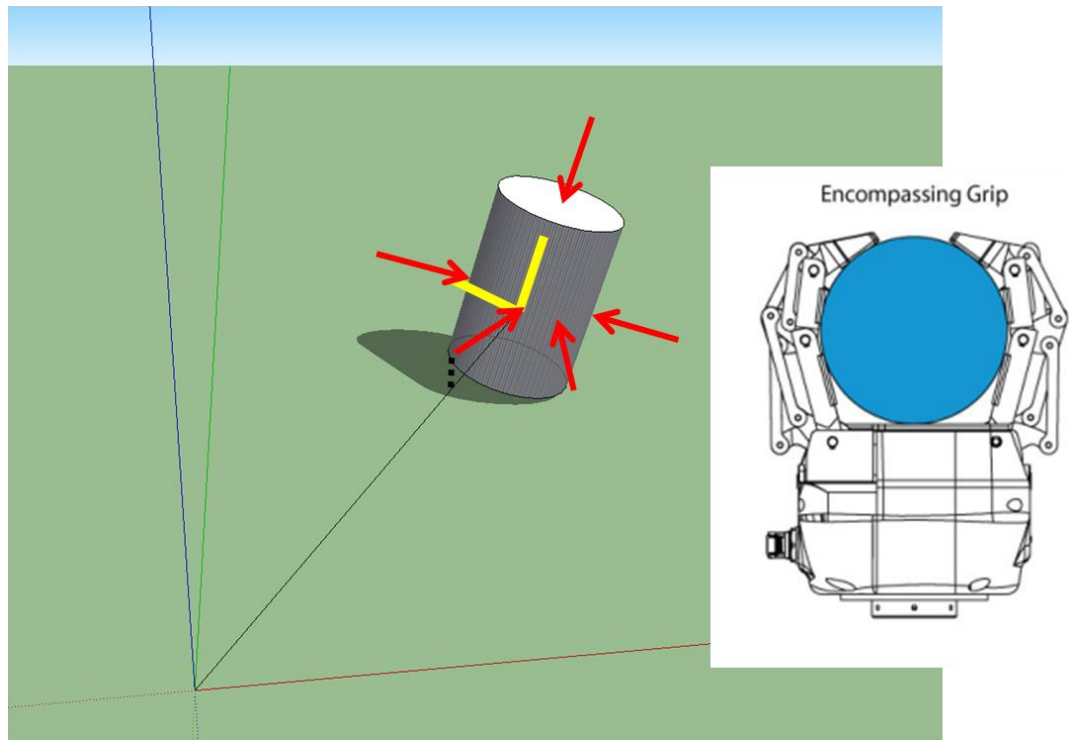


Figure 4.4: Grasp orientations about the sides and top of a cylinder object

Should inverse kinematic checks fail for either pose, the poses are recalculated by rotating about the center, vertical axis of the cylinder much like the bowl case while taking the cylinder radius into account. If none of the poses work, the final step is to try a different style of pinch mode grasp centered on the top of the cylinder. While such a pose would not be appropriate if contents of the cylinder needed to be poured out, the top grasp approach is suitable for simple container manipulation. The end effector pose is rotated to have the x-axis point downwards into the center, vertical axis of the can. The remaining axes again are constructed to be orthogonal to the surface normal of the top of the can, and the operational mode is set to pinch. The approach pose is formed by backing upwards along the x-axis of the gripper to provide clearance for the fingers. Should no grasp be possible at this final location, an error is returned.

#### **4.1.3.3 Box**

The box case lacks the advantage of rotational symmetry present in the other two primitives. However, feasible grasps are still possible to generate. Grasping strategy depends largely on the dimensions of the box. For small boxes it may be possible to use the basic operational mode or the pinch mode. Larger boxes are limited to the pinch mode if fingers cannot encompass the object. For a box with the longer dimension lying parallel against the horizontal plane of the world, a top down approach is used for the pinch mode. Again the vector from robot base to object is rotated so it is centered above the object and pointed downwards. This vector is used for the x-axis of the gripper frame and the other axes are constructed orthogonally so that the fingers pinch around the object. If a box is situated with the longest dimension vertically and the other dimensions are not too large for the reach of the fingers, the initial pose construction can be treated much like a cylinder. Instead of rotating the poses if inverse kinematic checks return errors, the poses are translated along the long dimension of the box primitive. Possible x-axes of the gripper for a box primitive are shown in Figure 4.5 (dimensions known from computer recognition). The box primitive type is not as common in the LANL dataset as the other two types, but encompasses a range of box types that have a dimension within the work range of the fingers which is up to about 16cm wide [Robotiq, 2013]. Objects larger than this dimension will not be able to be successfully grasped by the Robotiq.

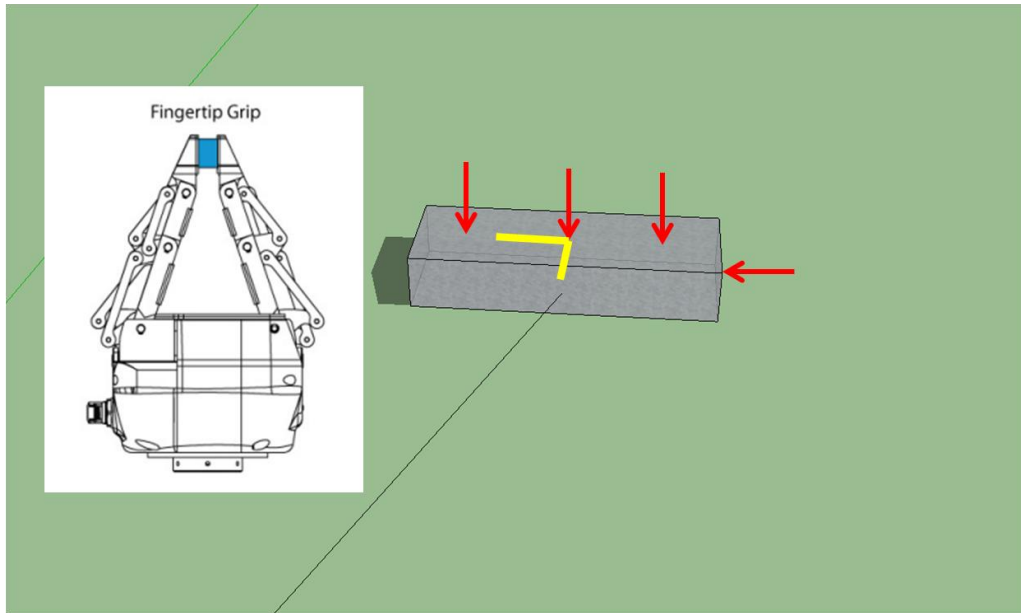


Figure 4.5: Grasp orientations about the sides of a box object

## 4.2 ALGORITHM REVIEW

This chapter described the bulk of the calculations and software structure involved in the grasp planning algorithm. The grasp planning algorithm is contained within a node in ROS. Information from the computer vision system is passed to this node and contains information about an object's identity and pose (position and orientation). Depending on the object's shape, the algorithm will attempt to construct two poses: an initial approach pose for the robot to move the gripper, allowing it space to open the gripper without collisions, and a secondary end effector pose for the gripper where finger closure and grasping will occur. Once these poses are calculated, it is verified that they are currently reachable according to the inverse kinematics of the robotic arm. If inverse kinematic calculations suggest those poses are unreachable, additional poses are calculated to search for a grasping solution. Demonstration of this algorithm at work is presented in the following chapter.



## **Chapter 5: Hardware, Demonstration, and Analysis**

In this chapter, the hardware and software architectures are outlined and the simulation and grasp plans are calculated on test cases to show algorithm functionality. These cases were picked because they cover the majority of items within the LANL dataset. Additionally, the rates of inverse kinematic check failures were analyzed as well as artificially simulated to test the algorithmic handling of invalid grasps.

### **5.1 HARDWARE**

The proposed grasping algorithm will be developed and evaluated based on the IRAD (Industrial Reconfigurable Anthropomorphic Dual-Arm) test bed at the University of Texas at Austin (Figure 5.1). The IRAD hardware components are summarized below. To create an algorithm which can be both accurately simulated and implemented, the same hardware is modeled when grasping is done in simulation.

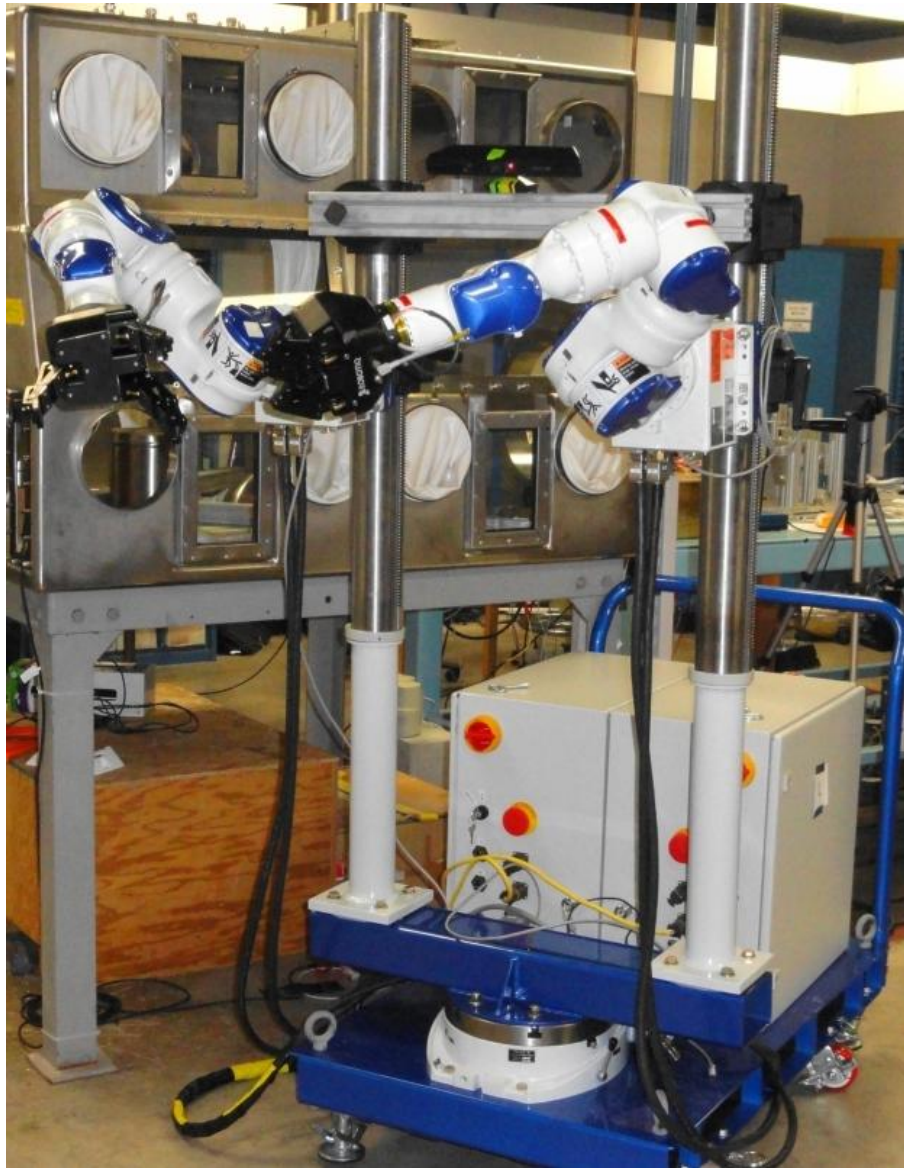


Figure 5.1 The IRAD dual arm system

### 5.1.1 Robotiq Gripper

Robotiq's Adaptive Gripper S Model has been identified as a suitable candidate for glovebox operations. It is a three fingered underactuated gripper. Each articulated finger has three joints (or three phalanges) depicted in Figure 5.2 [Robotiq, 2011]. For grasping an object, up to ten points of contact can be made between gripper and object:

three on each finger plus the base or “palm” of the hand. The underactuation made the Robotiq gripper a desirable candidate as it simplifies overall control. The construction of the fingers also contributed to the overall desirability of the gripper for glovebox environments. Some other robotic hands that have similar functions to the Robotiq are controlled via cables or “tendons” that actuate the fingers (such as the Meka H2 hand). While no component is immune to failure, cables tend to be an especially weak point on grippers that utilize them. The cables can snap and typically reduce the payload capacity. The Meka H2 has a maximum payload of 2.0 kg [Meka, 2009] while the Robotiq gripper has a payload ranging from 2.5 kg for a fingertip grasp to 10 kg for an encompassing grasp [Robotiq, 2013]. The robustness of the Robotiq gripper is due in part to its mechanical linkages for finger actuation instead of a cable-driven assembly.

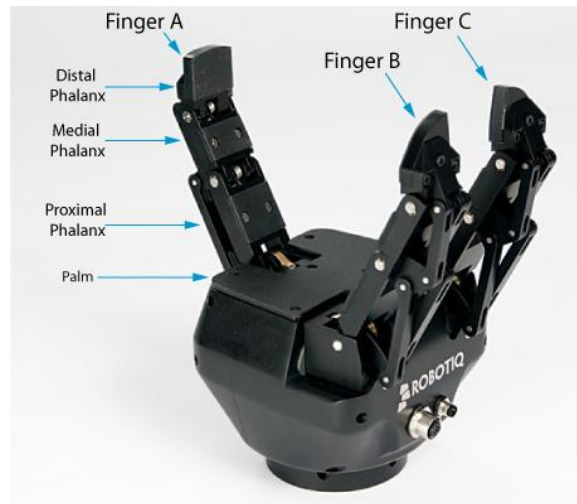


Figure 5.2: The Robotiq and its three fingers [Robotiq, 2011]

The Robotiq gripper is capable of two basic types of movements. One is a reorientation of Fingers B and C, shown in Figure 5.3, to one of four Operation Modes.

The Operation Modes serve to effectively change the shape of a grasp, and one is selected before a grasp is executed.

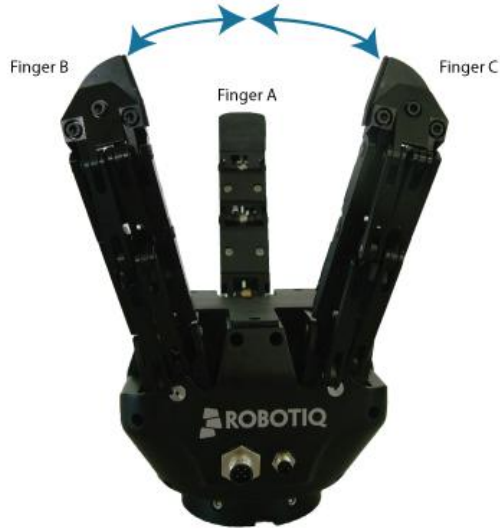


Figure 5.3: Variable finger movement for Fingers B and C [Robotiq, 2011]

The four Operation Modes are basic mode, wide mode, pinch mode, and scissor mode. The basic mode situates the spread of Fingers B and C at a medium distance and is a versatile, typically encompassing grasp type. The wide mode spreads Fingers B and C wider than the basic mode, and offers a similar grasp type that is suited for objects larger than ideal for the basic mode. Both the basic and wide Operation Modes will tend towards an encompassing grasp type, but can be used to precision grasp with the fingertips depending on grasped object geometry. The pinch mode closes the spread of Fingers B and C and pinches the fingertips of all fingers together in a precision grasp type. This mode is suitable for small objects. The final mode, scissor mode, uses the motion of Fingers B and C laterally together as the grasping motion to precision grasp small objects, but is not a very powerful grasp. Finger A is not used in scissor mode. The Operation Modes are shown in Figure 5.4.



Finger 5.4: Operational modes of the Robotiq [Robotiq, 2011]

The second type of movement available for the Robotiq gripper is the closing of the fingers, shown in Figure 5.5, which constitutes the maneuver for which most of the Operation Modes will come into contact with an object to be grasped. The closing of each finger is not controlled separately. Each finger will close until it reaches a stable configuration in contact with either an object or the palm of the gripper. The contact is detected when the current limit (determined by closing force) for each finger is exceeded. The force and speed at which the gripper closes is adjustable.

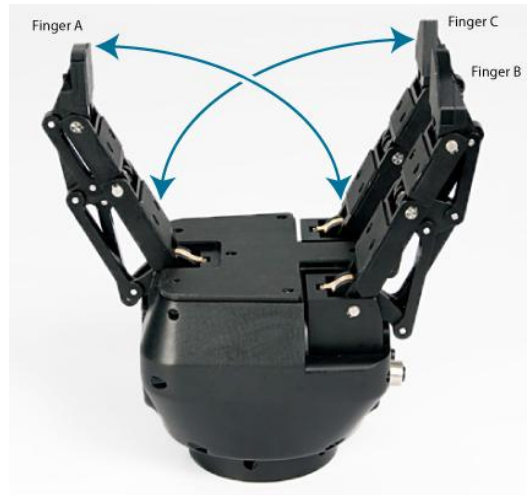


Figure 5.5: Finger closing movement for the Robotiq [Robotiq, 2011]

The Robotiq gripper is compatible with many different communication protocols. The protocol utilized in the Nuclear Robotics Group is Modbus TCP/IP. Modbus TCP/IP is built on the Modbus protocol developed in 1979 by Modicon. It is an open, industrial communication protocol that establishes server-client connections. Modbus TCP/IP uses Modbus with a TCP/IP wrapper [Modbus Organization, 2013]. The Robotiq gripper is connected via Ethernet cable to a router used for the greater robotic system.

In summary, the Robotiq gripper is highly suitable for a glovebox application. Its stronger-than-cable-driven mechanical linkages offer superior robustness when compared to other commercially available grippers. The Robotiq payload is compatible with and can even exceed that of the available payload of a Motoman SIA5D arm (5kg). The three fingers and their operational modes offer a high degree of adaptability to different grasping tasks, while the underactuation simplifies control. Additionally, while it is not explored in this work, a specialized glove for welding has been constructed for the Robotiq gripper [Robotiq, 2011]. This glove is not meant for radiological work, but its existence shows compatibility with gripper operation. This is significant should the

gripper itself be desired to be enclosed in a glove inside a glovebox to manage radiological contamination.

### **5.1.2 Motoman Arm**

The Nuclear Robotics Group IRAD uses two 7 DoF SIA5D robotic arms made by Yaskawa Motoman. The overall payload for an arm is 5 kg while the robot itself weighs 30 kg. The robot can be mounted in different configurations (floor, wall, or ceiling) and features an offset elbow joint to enlarge its workspace. An inner channel allows some wiring to be routed internally for peripherals mounted at the wrist of the robot [Yaskawa America, Inc., 2012].

The SIA5D shows promise for operating within a glovebox. The compact size allows it to be inserted through a standard gloveport. This port availability enhances the possibility of a glovebox deployed robot to replicate procedures that a human operator would perform without substantial alteration of the procedure itself – that is to say, the robot could potentially mimic a human in a glovebox so a redesign of the internal glovebox working environment may not be as extensive or even necessary. The SIA5D robots are shown mounted on a stand which enables their insertion into the glovebox in Figure 5.1.

### **5.1.3 Sensors**

The IRAD system is equipped with various other sensors to aid in its operation. An ATI Industrial Automation Gamma six-axis force torque sensor is mounted at the wrist of each robot. A Microsoft Kinect is employed as a computer vision device to capture information about objects within the workspace. While for simulation purposes the object information coming from the vision system is assumed, it is assumed to be the same type as in reality.

The assumed information from the vision system is built from previous work converting the depth information from the Kinect camera into a useful depiction of objects within a glovebox environment. A Euclidean clustering scheme is used to separate points in the depth information into clusters that represent objects. The clusters are then passed into a recognition system that uses a Cylindrical Projection Histogram to extract information about the cluster such as size, shape, and viewpoint. Using probabilistic object models, the extracted features of the clusters are trained on a dataset to identify an object and its position including pose [O’Neil, 2013]. The object and its pose is the key information that must be provided to the grasp planning algorithm.

#### **5.1.4 Hardware Summary**

The hardware described here is a system that has three main components: robotic hands attached to robotic arms with robotic eyes provided by the Kinect. While there are some ways in which the proposed grasping algorithm can be applied to any system, there are certain features and limitations of the hardware that drive parts of the algorithmic development. The structure of the hand drives a large part of the type of response available when faced with different shape primitives because the hand sets the bounds on the types of pregrasps available. The Robotiq’s four operational modes correspond to the pregrasps usable for the grasp planning algorithm. The dimensions of the hand also set limits on the size of objects that can be grasped (although items within the dataset introduced in the next section generally do not exceed these bounds). The workspace of the arm itself impacts the grasp planning algorithm in that it dictates where items must be located to be grasped. It is possible for an item to be located on the boundary of what is reachable so that some grasps may be valid whereas others are not. While the planning



algorithm itself is concerned about the interaction between object and hand, validation checks of a grasp plan ensure that the plan is not limited by the hardware.

## 5.2 SOFTWARE FRAMEWORK

The software framework builds off the systems described in Chapter Three, but here its specific implementation is detailed. The system software implementation uses ROS, Kinematix and AX capabilities. ROS is used primarily for sensor integration and trajectory planning, but the direct control of the SIA5D arm as well as additional safety features (such as collision detection) is handled by AX and Kinematix.

### 5.2.1 ROS

As previously described, ROS "programs" can be broken into a series of nodes that perform computations. The approach of using nodes is meant to reduce the code complexity by replacing a monolithic code structure with one that is modular. The grasping algorithm developed here is contained within a node identified as *grasp\_planner\_server*. The server designation comes from the ROS service communication paradigm which processes messages between nodes as a request/reply pair. The request for a grasp plan is sent by *grasp\_planner\_client*. The service message definition for grasp planning is shown in Figure 5.6.

```
int32 ObjectID
geometry_msgs/Pose poseObj
---
int32 success
geometry_msgs/Pose poseEEF
geometry_msgs/Pose poseAppr
string robotiqModel
```

Figure 5.6: Service message definition for grasp planning

The top portion of the service is the request portion. In making the request to the grasp planning server, the grasp planning client must send the item identification of what object is to be grasped (as an integer) as well as its pose (reflecting information that is provided by computer vision). The *geometry\_msgs/Pose* message, defined in a standard ROS package, has two parts - the x, y, and z coordinates corresponding to the position of the object and the x, y, z, and w values of the Quaternion which represent the object's orientation. The request data from the service definition gets sent to the grasp planner server when it is called. The grasp planner server then calculates a suitable grasp for the given object and returns appropriate poses for the robot to take for grasping. The returned values are an approach pose for the robot to move to for a pregrasp position, an end effector pose for the robot to move to for grasp execution, an integer representing what operational mode is required for the Robotiq gripper, and an integer that serves as a check for success or failure of the algorithm.

Within the grasp planning algorithm contained in the grasp planner server node, after the end effector poses have been calculated, the system checks if such a position is possible given the kinematics of the robot. Using MoveIt! the poses are tested for the existence of an inverse kinematic solution. The system checks for inverse kinematic solutions five times to account for the possibility of a false negative given the probabilistic nature of the solver in MoveIt!. If all checks against inverse kinematics show that such a pose is not feasible for the robot to reach, then the grasp planner will calculate new grasping poses and repeat the process.

For a fully integrated system with computer vision, the grasp planning would be only part of the full program. The grasp planning client node acts as a simplified surrogate for a node that would act as the main or base node of a full system. Such a node would take data from another node that operates the computer vision and pass on

visual information to the grasp planner algorithm. This main node would also be responsible for sending the computed end effector poses to the system which physically controls the robot arm and hand. A node to control communication between a computer and a Robotiq gripper is available for ROS as part of the ROS Industrial package. The node framework is shown in Figure 5.7.

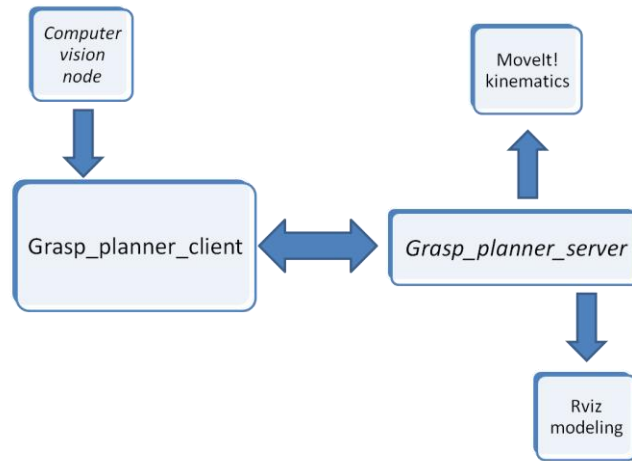


Figure 5.7: ROS grasp planning node structure

### 5.2.2 AX

The control of the Motoman arms is through the use of two AX controllers from Agile Planet. These controllers interface with computer system that is Windows 7-based and shares memory with a virtual machine running Windows CE. While motion planning can be carried out with this system, attempts to further ROS integration have enabled motion planning to be calculated on a separate ROS computer via MoveIt!. ROS calculated moves for the robot are sent via TCP/IP to the Windows system to send the commands to the controllers.

### 5.3 INITIAL SIMULATION SETUP

The simulations were performed using the Groovy distribution of ROS installed with the Ubuntu version 12.04 operating system. The ros node MoveIt! loads the layout of the robotic arm and the processes to calculate inverse kinematics. The grasp plan server is run to setup the node for grasp calculations, and the grasp plan client is setup to feed information into the grasp plan server. It is optional to run RViz as a simulation tool for the grasp planning portion of the code. If started, RViz will extract information from the grasp planning node to display the object and calculated grasping axes graphically. For these demonstration cases, bowl and canister types of primitives were used.

### 5.4 CANISTER

A cylinder of varying size was simulated to be located at various locations around the robotic arm. The cylinder radii used were 3cm and 6cm. The grasp planner successfully returned grasp plans for each cylinder as long as they were within the workspace of the robot. A typical output for the cylinder case is shown in Figure 5.8.

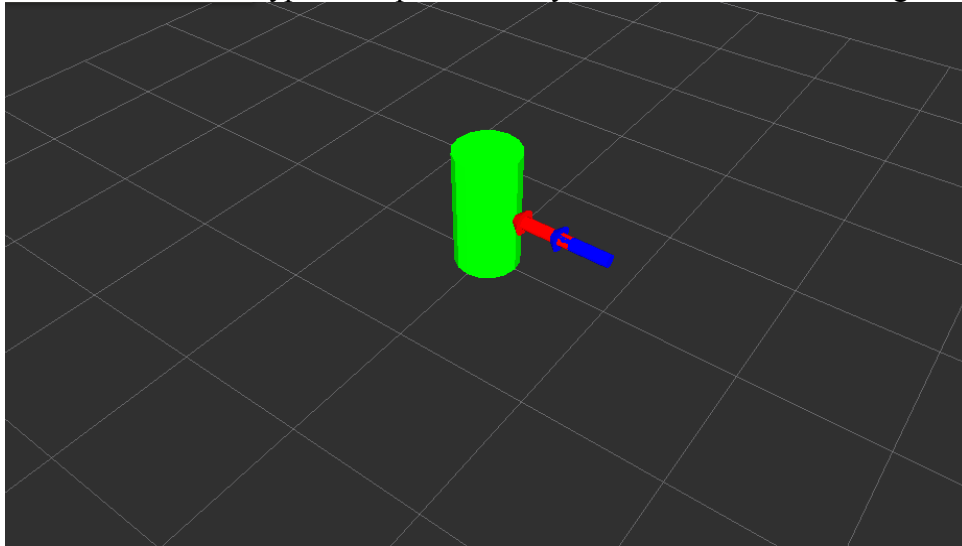


Figure 5.8: RViz output for a canister grasp plan

The red arrow represents the x-axis of the end effector pose of the gripper. The z-axis is perpendicular to the surface normal at this spot and towards the left of the image. The blue arrow is the approach pose. The majority of positions tested were able to return a successful grasp on the first attempt. A grasp plan output for the top of the can is shown in Figure 5.9.

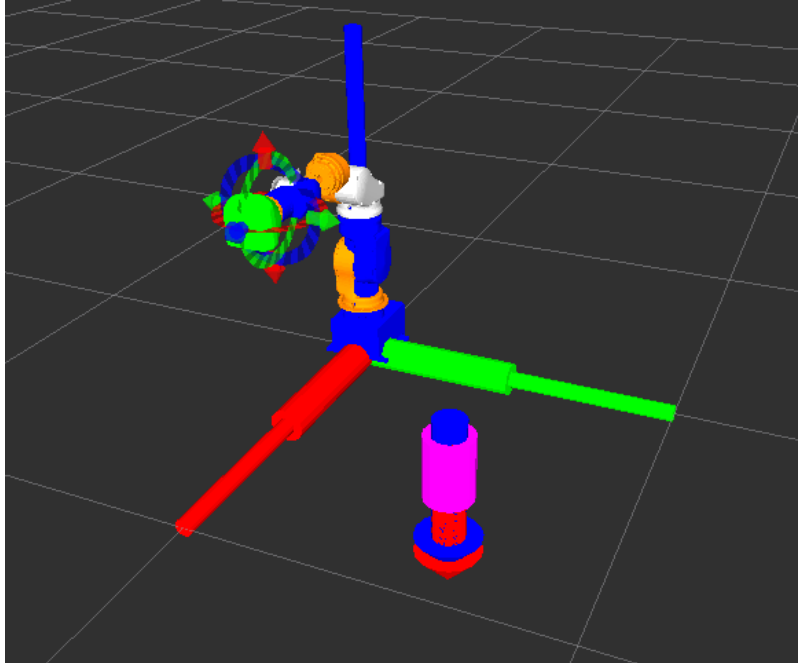


Figure 5.9: RViz output for a top down canister grasp plan

## 5.5 BOWL

A bowl of varying size was simulated at various locations around the robotic arm like the cylinder case. The bowl radii used were 6cm and 10cm. Again the grasp planner was successful in returning grasp plans along the lips of the bowls when the bowls were within the workspace of the robot. A typical output for the bowl case is shown in Figure 5.10.

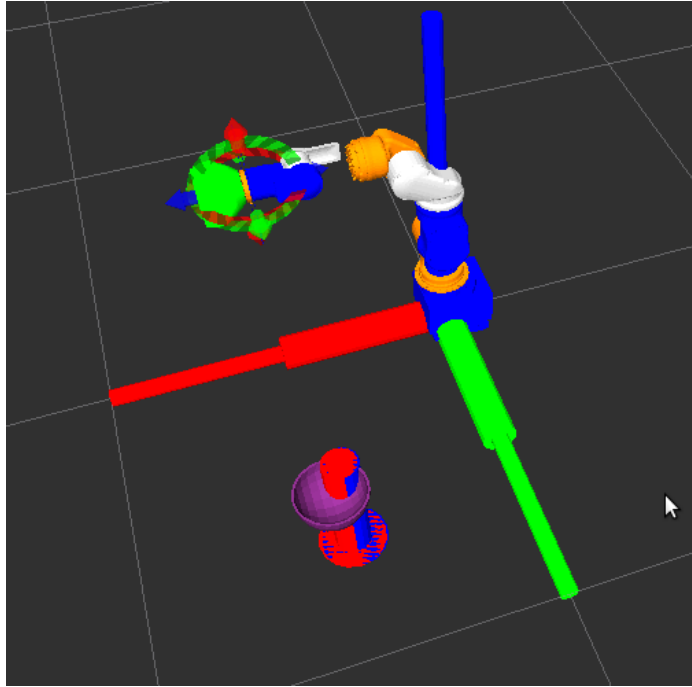


Figure 5.10: RViz output for a bowl grasp plan

The red arrow represents the x-axis of the end effector pose. The z-axis is pointed radially out from the central, vertical axis of symmetry for the bowl. The blue arrow is the approach pose.

## 5.6 INVERSE KINEMATIC FALSE NEGATIVES

The inverse kinematic solvers from MoveIt! are probabilistic in nature. It is therefore possible to get a check that returns an invalid grasping pose when it is indeed possible to reach that location with the robot. It is for this reason that inverse kinematic solution validity is checked five times, and the extra processing time is negligible. To analyze if the five checks are adequate, a series of grasp plans were constructed for a bowl type object one hundred times. In these one hundred tests, there were no instances of a valid object location that returned five inverse kinematic check failures. There was

one instance of a kinematic check that returned four failures. Over 95% of the results were either no kinematic check failures or only one check failure.

### **5.7 INVERSE KINEMATIC FALSE POSITIVES**

To test the performance of the error handling for when there is an inverse kinematic check failure, the code was modified to temporarily assume every calculated grasp plan was invalid according to current robot geometry. The program went through every step as normal, but a counter for inverse kinematics was always set to five failures after every calculation. This had the effect of forcing the program to calculate additional grasp plans until it could find no more. For each object type the algorithm successfully ran through additional grasp plans in the predetermined locations based on primitive type. Once it exhausted the list of possible grasp configurations without a successful plan, the program declared an error that the object in question was not currently reachable. For the bowl and cylinder test subjects, this means that the algorithm completed a complete revolution around the central axis of symmetry without finding a solution (in addition to the top side of the cylinder).

### **5.8 CANISTER WORKSPACE TEST**

The reach of the robot in the workspace in regards to cylinders was tested for the hardware in simulation. Cylinders were utilized as the shape primitive to assess the reach of a top down approach versus side grasps as two distinct cases. For the majority of the workspace for a 6cm cylinder, solutions are able to be found in either category. These were tested by running the algorithm in a grid of positions (spaced 5cm apart) around the robot. There are slight differences in the reachable cases for cylinders near the base of the robot and towards the outer edge of reach as seen in Figure 5.11. These are most likely due to the potentially shorter distance needed between robot and object for a side

grasp of a cylinder versus the top down approach, while the difficulties near the base of the robot come from limitations on maneuverability when the robot is operating near joint limits.

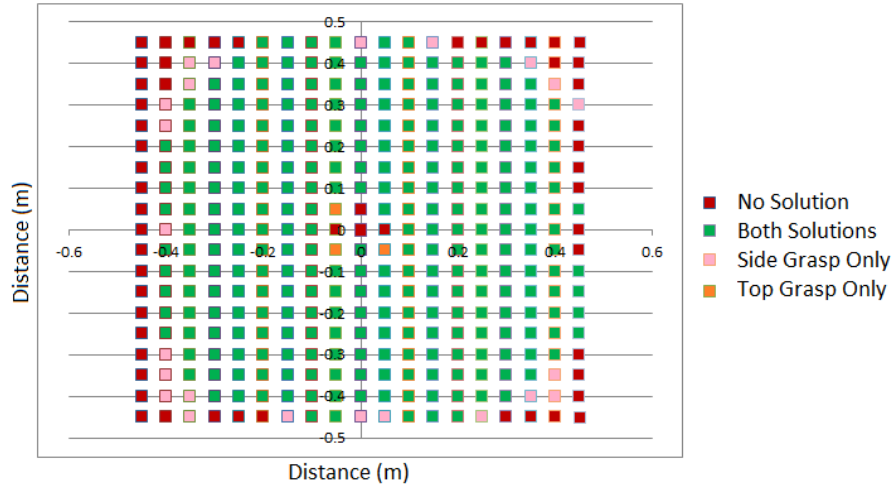


Figure 5.11: Workspace reach for cylinders

## 5.9 SUMMARY

This chapter described the hardware used to run tests of the grasping algorithm and basic protocols to run the grasp planning algorithm in ROS. The hardware modeled is the IRAD system at the University of Texas at Austin. While the hardware is a dual arm system, this work focuses on the planning for one arm which has the benefit of increasing algorithm portability. The grasp planning algorithm was tested and found successful in planning grasps that were able to be reached by the robotic arm. The algorithm's results can sometimes be unclear about the reason for an invalid grasp plan. There is currently no kinematic check in place to see if a grasp plan is failing just because an object is outside the robot's reachable workspace. In other words, the orientation of gripper and object may be feasible if there were less distance between robot and object. A possible shortcoming of the algorithm is not checking if an object is immediately



reachable because this would prevent the system from having to check all other permutations of available grasp plans before declaring an error. However, each grasp plan calculation is on the order of seconds or less and so the extra computational time is not extremely burdensome. The overall ability of the algorithm to generate initial and additional grasp plans is adequate for the objects seen here.

## Chapter 6: Conclusion

Grasping is a complicated problem in the robotics world. There are various ways to take advantage of a grasping situation to make this problem easier. As has been seen in this work, these ways range from environmental variables (e.g. glovebox restrictions) to gripper design (e.g. underactuation). While grasping is just one part of a robotics solution, it is important to the overall successful implementation of robotics for hazardous radiological operations. Robots must be proven to reliably and safely handle hazardous material if they are to replace humans in these tasks. Grasping is especially important because it is the physical point where the robot and material interact, and thus grasping reliability makes or breaks a successful robotic adoption.

This chapter summarizes the material from the previous chapters and offers suggestions for future research to improve upon the work presented here.

### 6.1 SUMMARY

**Chapter 1** laid the groundwork for the problem in grasping in a nuclear environment. Object-based grasping needs to, in general, know what the object is and where it is located and oriented in order to plan a grasp. The many types of robotic hands can be roughly classified into human and non-human inspired. Human hands are unparalleled in their dexterity, but robotic hands that mimic their structure are difficult to control. Nonanthropomorphic hands come in many forms, but the most dexterous ones are typically three fingered grippers. This class of grippers offers an optimal solution between dexterity and ease of control.

For this effort, we considered the Robotiq gripper because it has an ideal mix between robustness and dexterity. It can grasp a wide variety of objects while also lacking the cable-driven tendon systems, present in some anthropomorphic hands, which

can be prone to snap. The Robotiq thus may not be as prone to mechanical issues or malfunctions while in a glovebox. Due to access restrictions into a radiological glovebox, this is a key benefit. While underactuation simplifies hand control, it does provide the challenge of not being able to control precise fingertip locations for grasping. Based on this research, this control method appears suitable for glovebox work.

**Chapter 2** reviewed work in the field of grasping. Early work in grasping focused on the control of fully articulated robotic hands. Foundational research expanded the knowledge of the different functional types of grasps used to hold objects and how to measure the quality of such grasps. The invention of underactuated hands eased the overall control of these hands, but limited the fine control of finger placement. Strategies to calculate a grasp type based on the physical dimensions of the grasped object were explored. Based on this review, this research focused on addressing the critical challenge of identifying a suitable grasping pose for an array of objects that could be found in a glovebox. The review showed that it is difficult to predict the exact outcomes of a grasp for an underactuated hand, but the mechanism of finger closure should allow stable outcomes.

**Chapter 3** detailed the information that must be provided to a grasp planning algorithm. To grasp an item, one must have a general idea of what the item is and its spatial location and orientation relative to some known point. Technologies that can be used to provide this *a priori* information to the grasp planner were explored. While tactile and force feedback can provide useful information, computer vision is ideal due to the passive interaction with hazardous materials. Chapter 3 additionally outlined the operational software libraries available to incorporate all technologies needed to operate a robotic system.

**Chapter 4** defined the proposed grasping solution. The grasp planning algorithm is tuned to work with items from the LANL dataset of items identified as typical objects found in a glovebox environment. These items can be identified by a computer vision system which provides their position and orientation on a planar surface (such as the bottom floor of a glovebox). Using this information, the grasp planning algorithm can generate approach and end effector poses which represent the physical configuration of robotic arm and gripper together for grasping of an object. The initial approach direction is based on the vector from robot to object for simplicity, as well as the overall type of shape of the object. If either pose will not work according to the kinematics of the robot, then additional poses are calculated based on the geometry of an object until a solution is found.

**Chapter 5** described the hardware for implementation. Robotiq's Adaptive Gripper S Model is an ideal gripper for this application due to its three fingered dexterity and robustness. It has also been proven to work with gloves should that option be pursued in the future for glovebox work. The Yaskawa Motoman SIA5D robotic arm is a suitable candidate for glovebox work. It can be inserted into a glovebox through a gloveport. This hardware is part of the Industrial Reconfigurable Anthropomorphic Dual-Arm (IRAD) test bed at the University of Texas at Austin.

The operation of the grasping algorithm was demonstrated on test cases of a cylinder and bowl. It analyzed the solutions calculated for these cases and showed the algorithm was able to generate suitable grasps. In order to test the suitability of recalculations if a calculated pose is deemed unreachable, inverse kinematic checks were simulated as failures. The grasp planner was found to provide grasping poses that worked in a variety of object poses and reliably calculated additional poses when necessary due to robot kinematics.

## **6.2 FUTURE WORK**

There exists many ways in which this research can be expanded. The following areas are not exhaustive, but offer suggestions for key areas that would improve the capability of reliability of a robotic grasping system.

### **6.2.1 Deployment Research**

While much of this work has been conducted with the glovebox environment in mind, and the algorithm seems suitable for glovebox operation, there is work to do before this system can be implemented in an actual glovebox. The gripper hardware itself should be studied for its reaction to long term radiation exposure. It is known that radiation can have an effect on electrical components, and it should be better understood if radiation would pose a significant increase for gripper failure. Similarly, a failure mode and effects analysis should be completed for the hardware as other failures can occur beyond those possibly caused by radiation (e.g. a more mundanely-caused electrical failure). This knowledge is important because if the gripper is installed on the hot side of the glovebox and fails, it can pose a significant risk to a human operator to gain access to the inside to retrieve or fix the hardware. Frequent rates of failure would mean that the system would not be suitable for radiological glovebox work.

Work would also have to be done to assess the manner in which a robotic gripper would be incorporated into the airtight seal needed when working with SNM. One idea being explored in the NRG is to have the robotic arm enclosed in a sleeve similar to the gloves currently used by human operators. At the wrist location on the robot there could be a connection that would enable the robotic hand to be attached to the sleeve but on the hot side (whereas the rest of the robot remains on the cold side). Alternatively, given that the Robotiq has been gloved for other applications, the hand could be incorporated on the

cold side of the glovebox. These options need to be fully explored for a successful deployment.

It would also be beneficial for a statistical analysis of the grasp planner failures in a lifelike operating environment. This would be useful to determine the frequency of false positives generated by the grasp planning algorithm, if any, and the reasons for such failures. One possible failure mode of interest is what happens if the vision system falsely identifies an object to be of the wrong type. The resulting grasp would likely fail, but there may be sufficient similarities in the approach strategies for the underactuated fingers to provide a reliable grasp. Additional checks could be put in place to monitor for the anticipated force for a given grasp and abort if these are not observed.

### **6.2.2 Expanded Dataset**

Expanding this research to more items and thus more item shape types would increase the generality of such an algorithm. This work utilizes three basic shape primitives, but many more exist in the literature. These three shapes cover many of the general types of objects that are found in a glovebox, but additionally identifying more unique objects that exist in a glovebox would offer increased grasping capabilities. Options are to create more primitive types such as pyramids or toroids (could be used for a more specialized grasping plan for a roll of tape) or expand the supported unique object types. Many of the objects in the LANL dataset are metallic, but it is quite common for gloveboxes to be used for chemistry research; Chemistry equipment often consists of many types of glassware. While it may be more difficult to identify transparent objects with computer vision, improved grasping for objects such as test tubes and flasks would enlarge the scope of a robotic glovebox system for LANL.

### **6.2.3 Task Heuristics**

The possibility exists for better task support to be taken into account for the grasping algorithm. Cylindrical canister objects are initially preferred to be grasped from the side of the canister in case it is desired to have the canister's contents poured out. Implementing a system to recognize and learn the types of tasks commonly associated with a particular object type could lead to better support of glovebox operations. For example, should a larger canister type be typically lowered into a receptacle whereas a smaller canister type is typically used to pour its contents into another container, different initial grasping strategies would be ideal for each type. The large canister would be better suited to a top down grasp style while the small canister would be best with the aforementioned side grasp. However, glovebox operations are not always well defined publicly, so a runtime heuristic system could provide better customization while maintaining generality of the algorithm for multiple gloveboxes.

### **6.2.4 Grasp Optimization**

The goal of this grasping algorithm is to provide a stable grasp that will work given the current configuration of object and robot together. It does not try to find an optimal grasp. Future work could expand upon this work by using the algorithm to provide a set of stable grasps. From this set of grasps the best grasp could be selected for use using various grasp metrics as described in Chapter 2. Overall processing time could potentially be increased with generating a full set of grasps, but this approach could generate grasps that are finer tuned for the object at hand.

### **6.3 CONCLUDING COMMENTS**

There exists a great opportunity in the nuclear complex to modernize technology used for handling of hazardous materials. There have been many attempts in the past to create improvements in the nuclear domain using robotics, but multitudes have ultimately failed to be adopted. The reasons for the failure or success of proposed robotics systems is deserving of more study, but one key issue addressed in this project is the tedious nature of grasping during teleoperation and or the manual grasp planning for autonomous processes. The planner developed through this research can be used in either situation to reduce the burden on the operator and/or system designer.

Some autonomous systems have also been stalled due to budgetary concerns, but this could be primarily a lack of priority for robotic replacements compared to other expenses. The inclusion of a grasp planner is one small part of a larger system that can reduce cost by reducing integration time and trial & error methodologies for grasp planning that are common today. While no robotic system may be perfect, the potential improvements to human safety and security that can be provided by robotics is worth a closer look, especially when faced with aging radiological facilities that will need to be either upgraded or replaced in the coming decades.



## Appendix – Source Code and Usage

This appendix contains source code relevant to the grasp planning algorithm as described in Chapter Four. It is simplified code from within the *grasp planner server* node within the ROS structure showing examples of the three basic primitive types. This code was developed using the ROS Groovy Galapagos distribution release on computers running Ubuntu 12.04 LTS. Many excellent resources exist online to guide a user through the installation of ROS such as the information that can be found at the main ROS website (<http://wiki.ros.org/>). Additional packages are needed namely MoveIt!, TF, Eigen, and visualization\_msgs. ROS includes a prerequisite collection of nodes and programs called *roscore* that must be run prior to launching additional ROS programs such as the one below. The following node can be communicated with via the service message definition in Figure 5.6.

```
#include "ros/ros.h"
#include "grasp_planner/grasp_plan.h"
#include <stdio.h>
#include <tf/tf.h>
#include <Eigen/Core>
#include <Eigen/Geometry>
#include <visualization_msgs/Marker.h>
#include <tf/transform_broadcaster.h>
#include <moveit_msgs/GetPositionIK.h>
#include <moveit/robot_model_loader/robot_model_loader.h>
#include <moveit/robot_model/robot_model.h>
#include <moveit/robot_state/robot_state.h>
#include <moveit/robot_state/joint_state_group.h>
#include <moveit/robot_state/conversions.h>
#include <moveit_msgs/DisplayRobotState.h>

#define PI 3.14159

visualization_msgs::Marker marker;
visualization_msgs::Marker mrkEef;
visualization_msgs::Marker mrkAppr;

tf::Transform transEef;
tf::Transform transAppr;
tf::Transform transWorld;

//setup for IK checks
ros::ServiceClient ik_client;
moveit_msgs::GetPositionIK ik_srv_;           //service request for computing IK

//*****
//call back function for grasp planning
//this function contains the bulk of the grasping calculations
//it sets up initial variables for calculations and inverse kinematics
```

```

//then switches the calculations based on what object was identified using the
//vision system
//if the initial grasp does not meet IK checks, then the system will recompute
//until a solution is found or all options are exhausted
//*****
bool grasp_plan_cb(grasp_planner::grasp_plan::Request &req,
grasp_planner::grasp_plan::Response &res)
{
    //std::cout << "Pong\n";
    res.success = 1;

    //set marker object Pose equal to the object Pose
    marker.pose.position.x = req.poseObj.position.x;
    marker.pose.position.y = req.poseObj.position.y;
    marker.pose.position.z = req.poseObj.position.z;
    marker.pose.orientation.x = req.poseObj.orientation.x;
    marker.pose.orientation.y = req.poseObj.orientation.y;
    marker.pose.orientation.z = req.poseObj.orientation.z;
    marker.pose.orientation.w = req.poseObj.orientation.w;
    //marker.scale.x = 1.0;
    //marker.scale.y = 1.0;
    //marker.scale.z = 1.0;
    //setting color and visibility
    marker.color.r = 1.0f;
    marker.color.b = 1.0f;
    marker.color.g = 0.0f;
    marker.color.a = 1.0;

    //Eigen variables
    Eigen::Quaternionf eefQuat;
    Eigen::Quaternionf apprQuat;
    Eigen::Quaternionf objQuat;
    Eigen::Quaternionf invQuat;
    Eigen::Quaternionf rvizQuat;

    Eigen::Matrix3f objRot;
    Eigen::Matrix3f tmpRot;
    Eigen::Matrix3f tmpRot2;

    double theta = 0;
    int errCount = 0;

    //initializing object Quaternion to request object quaternion and
    normalizing
    objQuat.x() = req.poseObj.orientation.x;
    objQuat.y() = req.poseObj.orientation.y;
    objQuat.z() = req.poseObj.orientation.z;
    objQuat.w() = req.poseObj.orientation.w;
    objQuat.normalize();
    objRot = objQuat;
    //std::cout << objQuat.x() << "\n" << objQuat.y() << "\n" << objQuat.z() <<
    "\n" << objQuat.w() << "\n";

```

```

//Eigen vector variables
Eigen::Vector3f objArrow;
Eigen::Vector3f unitArrow;
Eigen::Vector3f eefVec;
Eigen::Vector3f apprVec;
Eigen::Vector3f dotVec;
Eigen::Vector3f edgeVec;
Eigen::Vector3f clrVec;
Eigen::Vector3f axisVec;

//create arrow from base from to object position
objArrow.x() = req.poseObj.position.x;
objArrow.y() = req.poseObj.position.y;
objArrow.z() = req.poseObj.position.z;
unitArrow = objArrow;
unitArrow.normalize();

//variables representing offsets between edges and kinect identified
position
Eigen::Vector3f canOffsets;
Eigen::Vector3f bowlOffsets;
Eigen::Vector3f boxOffsets;

//move it variables for IK checking
moveit_msgs::DisplayRobotState state_msg;
robot_state::JointStateGroup* joint_state_group;

//Set seed state to current state - for IK checking
robot_model_loader::RobotModelLoader
robot_model_loader("robot_description");
robot_model::RobotModelPtr kinematic_model =
robot_model_loader.getModel();
robot_state::RobotStatePtr kinematic_state(new
robot_state::RobotState(kinematic_model));
joint_state_group = kinematic_state->getJointStateGroup("sia5d");
ik_srv_.request.ik_request.robot_state.joint_state.name =
joint_state_group->getJointModelGroup()->getJointModelNames();
joint_state_group->setToRandomValues();
joint_state_group-
>getVariableValues(ik_srv_.request.ik_request.robot_state.joint_state.position);

ik_srv_.request.ik_request.avoid_collisions = true;

//from here on code depends on object type

// look up object ID
//switch based method for now
switch (req.objectID){

    case 1 : // can
//statements

```

```

std::cout << "Case 1\n";

canOffsets.x() = 0.00;
canOffsets.y() = 0.00;
canOffsets.z() = 0.10;

//move arrow towards desired point of contact in object frame
objArrow = objArrow + objRot*canOffsets;
eefQuat = objRot;
apprQuat = objRot;

canOffsets.x() = 0.05;
dotVec = (-objArrow.dot(objRot.col(0)))*objRot.col(0);
dotVec.normalize();
edgeVec = canOffsets.x()*dotVec;

res.poseEEF.position.x = objArrow.x()+edgeVec.x();
res.poseEEF.position.y = objArrow.y()+edgeVec.y();
res.poseEEF.position.z = objArrow.z()+edgeVec.z();

clrVec = 0.1016*edgeVec;
res.poseAppr.position.x = objArrow.x()+edgeVec.x()+clrVec.x();
res.poseAppr.position.y = objArrow.y()+edgeVec.y()+clrVec.y();
res.poseAppr.position.z = objArrow.z()+edgeVec.z()+clrVec.z();

axisVec = -edgeVec.cross(objRot.col(2));
tmpRot.col(0) = axisVec;
tmpRot.col(1) = -edgeVec;
tmpRot.col(2) = objRot.col(2);

eefQuat = tmpRot;
apprQuat = tmpRot;

res.poseEEF.orientation.x = eefQuat.x();
res.poseEEF.orientation.y = eefQuat.y();
res.poseEEF.orientation.z = eefQuat.z();
res.poseEEF.orientation.w = eefQuat.w();

res.poseAppr.orientation.x = apprQuat.x();
res.poseAppr.orientation.y = apprQuat.y();
res.poseAppr.orientation.z = apprQuat.z();
res.poseAppr.orientation.w = apprQuat.w();

mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0;

mrkAppr.pose.position.x = res.poseAppr.position.x;

```

```

    mrkAppr.pose.position.y = res.poseAppr.position.y;
    mrkAppr.pose.position.z = res.poseAppr.position.z;
    mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
    mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
    mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;
    mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
    mrkAppr.color.a = 1.0;

    //set the transform for EEF coordinate frames
    transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
    res.poseEEF.position.y, res.poseEEF.position.z));
    transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
    res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
    //br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
    "world", "EEF"));

    //set the transform for APPR coordinate frames
    transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
    res.poseAppr.position.y, res.poseAppr.position.z));
    transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
    res.poseAppr.orientation.y, res.poseAppr.orientation.z,
    res.poseAppr.orientation.w));

    //settings for the marker to represent the approach pose
    //blue
    mrkAppr.pose.position.x = res.poseAppr.position.x;
    mrkAppr.pose.position.y = res.poseAppr.position.y;
    mrkAppr.pose.position.z = res.poseAppr.position.z;
    mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
    mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
    mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;
    mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
    mrkEef.color.a = 1.0;      //make visible

    //std::cout << "Debug\n";

    //load the approach pose and check against IK
    ik_srv_.request.ik_request.group_name = "sia5d";
    ik_srv_.request.ik_request.pose_stamped.header.frame_id = "world";
    ik_srv_.request.ik_request.pose_stamped.pose.position.x =
    res.poseAppr.position.x;
    ik_srv_.request.ik_request.pose_stamped.pose.position.y =
    res.poseAppr.position.y;
    ik_srv_.request.ik_request.pose_stamped.pose.position.z =
    res.poseAppr.position.z;
    ik_srv_.request.ik_request.pose_stamped.pose.orientation.x =
    res.poseAppr.orientation.x;
    ik_srv_.request.ik_request.pose_stamped.pose.orientation.y =
    res.poseAppr.orientation.y;
    ik_srv_.request.ik_request.pose_stamped.pose.orientation.z =
    res.poseAppr.orientation.z;
    ik_srv_.request.ik_request.pose_stamped.pose.orientation.w =
    res.poseAppr.orientation.w;

```



```

//*****repeat process for grasp generation with new coordinates*****
//angle of base rotation
theta = atan2(objArrow.y(), objArrow.x());

//rotation matrices to orient hand in proper position for grasping
bowl lip
tmpRot << cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0,
0, 1;
tmpRot2 << cos(PI/2), 0, sin(PI/2), 0, 1, 0, -sin(PI/2), 0,
cos(PI/2);

//construct rotation matrix and set result quaternions to match
tmpRot = tmpRot*tmpRot2;
eefQuat = tmpRot;
apprQuat = tmpRot;

//load the EEF pose orientation
res.poseEEF.orientation.x = eefQuat.x();
res.poseEEF.orientation.y = eefQuat.y();
res.poseEEF.orientation.z = eefQuat.z();
res.poseEEF.orientation.w = eefQuat.w();

//load the APPR pose orientation
res.poseAppr.orientation.x = apprQuat.x();
res.poseAppr.orientation.y = apprQuat.y();
res.poseAppr.orientation.z = apprQuat.z();
res.poseAppr.orientation.w = apprQuat.w();

//settings for the marker to represent the EEF pose
//red
mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0; //make visible

//set the transform for EEF coordinate frames
transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));
transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
//br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

//set the transform for APPR coordinate frames
transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

```





```

        break; //if not equal to 5, then we have a good IK soln
and can stop trying
    }

    }// end while loop for errCount==5

    //try top
    errCount=0; //reset counter

    objArrow.x() = objArrow.x() + unitArrow.x()*0.03;
    objArrow.y() = objArrow.y() + unitArrow.y()*0.03;
    //angle of base rotation
    theta = atan2(objArrow.y(), objArrow.x());

    //rotation matrices to orient hand in proper position for top
    tmpRot << cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0,
0, 1;
    tmpRot2 << cos(PI/2), 0, sin(PI/2), 0, 1, 0, -sin(PI/2), 0,
cos(PI/2);
    //construct rotation matrix and set result quaternions to match
    tmpRot = tmpRot*tmpRot2;
    eefQuat = tmpRot;
    apprQuat = tmpRot;

    //load the EEF pose orientation
    res.poseEEF.orientation.x = eefQuat.x();
    res.poseEEF.orientation.y = eefQuat.y();
    res.poseEEF.orientation.z = eefQuat.z();
    res.poseEEF.orientation.w = eefQuat.w();

    //load the APPR pose orientation
    res.poseAppr.orientation.x = apprQuat.x();
    res.poseAppr.orientation.y = apprQuat.y();
    res.poseAppr.orientation.z = apprQuat.z();
    res.poseAppr.orientation.w = apprQuat.w();

    //settings for the marker to represent the EEF pose
    //red
    mrkEef.pose.position.x = res.poseEEF.position.x;
    mrkEef.pose.position.y = res.poseEEF.position.y;
    mrkEef.pose.position.z = res.poseEEF.position.z;
    mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
    mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
    mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
    mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
    mrkEef.color.a = 1.0; //make visible

    //set the transform for EEF coordinate frames
    transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));

```

```

        transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
        //br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

        //set the transform for APPR coordinate frames
        transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
        transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

        //settings for the marker to represent the approach pose
        //blue
        mrkAppr.pose.position.x = res.poseAppr.position.x;
        mrkAppr.pose.position.y = res.poseAppr.position.y;
        mrkAppr.pose.position.z = res.poseAppr.position.z;
        mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
        mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
        mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;
        mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
        mrkEef.color.a = 1.0;        //make visible

        //std::cout << "Debug\n";

        //load the approach pose and check against IK
        ik_srv_.request.ik_request.group_name = "sia5d";
        ik_srv_.request.ik_request.pose_stamped.header.frame_id = "world";
        ik_srv_.request.ik_request.pose_stamped.pose.position.x =
res.poseAppr.position.x;
        ik_srv_.request.ik_request.pose_stamped.pose.position.y =
res.poseAppr.position.y;
        ik_srv_.request.ik_request.pose_stamped.pose.position.z =
res.poseAppr.position.z;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.x =
res.poseAppr.orientation.x;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.y =
res.poseAppr.orientation.y;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.z =
res.poseAppr.orientation.z;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.w =
res.poseAppr.orientation.w;

        //reset errCount for IK checking
        errCount = 0;

        //Compute IK 5 times:
        for (int i=0; i<5; i++){
            if (ik_client.call(ik_srv_))
            {
                if(ik_srv_.response.error_code.val > 0)
                {

```

```

        std::cout << "IK solution found!" << std::endl;
    }else{
        std::cout << "IK failure: error_code: " <<
ik_srv_.response.error_code.val << std::endl;
        errCount += 1;
    }

    }else{
        ROS_ERROR("Failed to call ik service");
        res.success = 0;
        return false;
    }

    }// end for loop
    std::cout << "IK Errors: " << errCount << std::endl;

    }// end if loop for errCount==5
    if(errCount==5){
        std::cout << "No valid grasps found\n";
    }

    break;

    case 2 :        //box
        std::cout << "Case 2\n";
        //statements
        boxOffsets.x() = 0.00;
        boxOffsets.y() = 0.00;
        boxOffsets.z() = 0.02;
        objArrow = objArrow + objRot*boxOffsets;
        eefQuat = objRot;
        apprQuat = objRot;

        boxOffsets.x() = 0.02;
        dotVec = (-objArrow.dot(objRot.col(0)))*objRot.col(0);
        dotVec.normalize();
        edgeVec = boxOffsets.x()*dotVec;

        res.poseEEF.position.x = objArrow.x()+edgeVec.x();
        res.poseEEF.position.y = objArrow.y()+edgeVec.y();
        res.poseEEF.position.z = objArrow.z()+edgeVec.z();
        res.poseAppr.position.x = objArrow.x();
        res.poseAppr.position.y = objArrow.y();
        res.poseAppr.position.z = objArrow.z() + 0.10;

        //angle of base rotation
        theta = atan2(objArrow.y(), objArrow.x());

        //rotation matrices to orient hand in proper position for box top

        tmpRot << cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0,
0, 1;

```

```

cos(PI/2);

tmpRot2 << cos(PI/2), 0, sin(PI/2), 0, 1, 0, -sin(PI/2), 0,

//construct rotation matrix and set result quaternions to match
tmpRot = tmpRot*tmpRot2;
eefQuat = tmpRot;
apprQuat = tmpRot;

//load the EEF pose orientation
res.poseEEF.orientation.x = eefQuat.x();
res.poseEEF.orientation.y = eefQuat.y();
res.poseEEF.orientation.z = eefQuat.z();
res.poseEEF.orientation.w = eefQuat.w();

//load the APPR pose orientation
res.poseAppr.orientation.x = apprQuat.x();
res.poseAppr.orientation.y = apprQuat.y();
res.poseAppr.orientation.z = apprQuat.z();
res.poseAppr.orientation.w = apprQuat.w();

//settings for the marker to represent the EEF pose
//red
mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0; //make visible

//set the transform for EEF coordinate frames
transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));
transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
//br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

//set the transform for APPR coordinate frames
transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

//settings for the marker to represent the approach pose
//blue
mrkAppr.pose.position.x = res.poseAppr.position.x;
mrkAppr.pose.position.y = res.poseAppr.position.y;
mrkAppr.pose.position.z = res.poseAppr.position.z;
mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;

```

```

mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
mrkEef.color.a = 1.0;          //make visible

//std::cout << "Debug\n";

//load the approach pose and check against IK
ik_srv_.request.ik_request.group_name = "sia5d";
ik_srv_.request.ik_request.pose_stamped.header.frame_id = "world";
ik_srv_.request.ik_request.pose_stamped.pose.position.x =
res.poseAppr.position.x;
ik_srv_.request.ik_request.pose_stamped.pose.position.y =
res.poseAppr.position.y;
ik_srv_.request.ik_request.pose_stamped.pose.position.z =
res.poseAppr.position.z;
ik_srv_.request.ik_request.pose_stamped.pose.orientation.x =
res.poseAppr.orientation.x;
ik_srv_.request.ik_request.pose_stamped.pose.orientation.y =
res.poseAppr.orientation.y;
ik_srv_.request.ik_request.pose_stamped.pose.orientation.z =
res.poseAppr.orientation.z;
ik_srv_.request.ik_request.pose_stamped.pose.orientation.w =
res.poseAppr.orientation.w;

//reset errCount for IK checking
errCount = 0;

//Compute IK 5 times:
for (int i=0; i<5; i++){
if (ik_client.call(ik_srv_))
{
    if(ik_srv_.response.error_code.val > 0)
    {
        std::cout << "IK solution found!" << std::endl;
    }else{
        std::cout << "IK failure: error_code: " <<
ik_srv_.response.error_code.val << std::endl;
        errCount += 1;
    }

}

}

}else{
    ROS_ERROR("Failed to call ik service");
    res.success = 0;
    return false;
}

}

}

// end for loop
std::cout << "IK Errors: " << errCount << std::endl;

//check if there was a positive solution
if (errCount == 5){ //if 5 then no solution was found

//while (errCount==5){          //while there is not IK solution

```

```

for (int j=1; j<7;j++){ //changing to for loop

errCount=0; //reset counter

//try another position
objArrow.y() = objArrow.y() + edgeVec.y()*3 - (j-1)*0.02;
res.poseEEF.position.x = objArrow.x();
res.poseEEF.position.y = objArrow.y();
res.poseEEF.position.z = objArrow.z();
res.poseAppr.position.x = objArrow.x();
res.poseAppr.position.y = objArrow.y();
res.poseAppr.position.z = objArrow.z()+0.1;
//settings for the marker to represent the EEF pose
//red
mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0; //make visible

//set the transform for EEF coordinate frames
transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));
transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
//br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

//set the transform for APPR coordinate frames
transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

//settings for the marker to represent the approach pose
//blue
mrkAppr.pose.position.x = res.poseAppr.position.x;
mrkAppr.pose.position.y = res.poseAppr.position.y;
mrkAppr.pose.position.z = res.poseAppr.position.z;
mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;
mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
mrkEef.color.a = 1.0; //make visible

//load the approach pose and check against IK
ik_srv_.request.ik_request.group_name = "sia5d";
ik_srv_.request.ik_request.pose_stamped.header.frame_id = "world";
ik_srv_.request.ik_request.pose_stamped.pose.position.x =
res.poseAppr.position.x;

```



```

//offsets from camera position to bowl lip
bowlOffsets.x() = 0.0;
bowlOffsets.y() = 0.0;
bowlOffsets.z() = 0.03;
objArrow.z() = objArrow.z() + bowlOffsets.z();
res.poseEEF.position.x = objArrow.x();
res.poseEEF.position.y = objArrow.y();
res.poseEEF.position.z = objArrow.z() + 0.05;
res.poseAppr.position.x = objArrow.x();
res.poseAppr.position.y = objArrow.y();
res.poseAppr.position.z = objArrow.z() + 0.10;

//angle of base rotation
theta = atan2(objArrow.y(), objArrow.x());

//rotation matrices to orient hand in proper position for grasping
bowl lip
0, 1;
cos(PI/2);

tmpRot << cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0,
tmpRot2 << cos(PI/2), 0, sin(PI/2), 0, 1, 0, -sin(PI/2), 0,

//construct rotation matrix and set result quaternions to match
tmpRot = tmpRot*tmpRot2;
eefQuat = tmpRot;
apprQuat = tmpRot;

//load the EEF pose orientation
res.poseEEF.orientation.x = eefQuat.x();
res.poseEEF.orientation.y = eefQuat.y();
res.poseEEF.orientation.z = eefQuat.z();
res.poseEEF.orientation.w = eefQuat.w();

//load the APPR pose orientation
res.poseAppr.orientation.x = apprQuat.x();
res.poseAppr.orientation.y = apprQuat.y();
res.poseAppr.orientation.z = apprQuat.z();
res.poseAppr.orientation.w = apprQuat.w();

//settings for the marker to represent the EEF pose
//red
mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0; //make visible

//set the transform for EEF coordinate frames
transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));

```



```

        transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
        //br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

        //set the transform for APPR coordinate frames
        transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
        transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

        //settings for the marker to represent the approach pose
        //blue
        mrkAppr.pose.position.x = res.poseAppr.position.x;
        mrkAppr.pose.position.y = res.poseAppr.position.y;
        mrkAppr.pose.position.z = res.poseAppr.position.z;
        mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;
        mrkAppr.pose.orientation.y = res.poseAppr.orientation.y;
        mrkAppr.pose.orientation.z = res.poseAppr.orientation.z;
        mrkAppr.pose.orientation.w = res.poseAppr.orientation.w;
        mrkEef.color.a = 1.0;        //make visible

        //std::cout << "Debug\n";

        //load the approach pose and check against IK
        ik_srv_.request.ik_request.group_name = "sia5d";
        ik_srv_.request.ik_request.pose_stamped.header.frame_id = "world";
        ik_srv_.request.ik_request.pose_stamped.pose.position.x =
res.poseAppr.position.x;
        ik_srv_.request.ik_request.pose_stamped.pose.position.y =
res.poseAppr.position.y;
        ik_srv_.request.ik_request.pose_stamped.pose.position.z =
res.poseAppr.position.z;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.x =
res.poseAppr.orientation.x;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.y =
res.poseAppr.orientation.y;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.z =
res.poseAppr.orientation.z;
        ik_srv_.request.ik_request.pose_stamped.pose.orientation.w =
res.poseAppr.orientation.w;

        //reset errCount for IK checking
        errCount = 0;

        //Compute IK 5 times:
        for (int i=0; i<5; i++){
            if (ik_client.call(ik_srv_))
            {
                if(ik_srv_.response.error_code.val > 0)
                {

```

```

        std::cout << "IK solution found!" << std::endl;
    }else{
        std::cout << "IK failure: error_code: " <<
ik_srv_.response.error_code.val << std::endl;
        errCount += 1;
    }

    }else{
        ROS_ERROR("Failed to call ik service");
        res.success = 0;
        return false;
    }

    }// end for loop
    std::cout << "IK Errors: " << errCount << std::endl;

    //debug statement
    //errCount = 5;

    //check if there was a positive solution
    if (errCount == 5){ //if 5 then no solution was found

        //while (errCount==5){ //while there is not IK solution
        for (int j=1; j<13;j++){ //changing to for loop

            errCount=0; //reset counter

            //try another position

            //move point to center - in the objArrow direction
            objArrow.x() = objArrow.x() + unitArrow.x()*0.05; //adjust value
based on diameter of bowl
            objArrow.y() = objArrow.y() + unitArrow.y()*0.05; //adjust

            //rotate about z direction by about 28.6 degrees AKA 0.5 radians -
12 times
            tmpRot << cos(j*0.5),-sin(j*0.5), 0, sin(j*0.5), cos(j*0.5), 0, 0,
0, 1;
            objArrow = tmpRot*objArrow;

            //move back out to edge of bowl
            unitArrow = objArrow;
            unitArrow.normalize();
            objArrow.x() = objArrow.x() - unitArrow.x()*0.05; //adjust
            objArrow.y() = objArrow.y() - unitArrow.y()*0.05; //adjust

            //*****repeat process for grasp generation with new coordinates*****
            //angle of base rotation
            theta = atan2(objArrow.y(), objArrow.x());

            //rotation matrices to orient hand in proper position for grasping
bowl lip

```

```

0, 1;
tmpRot << cos(theta), -sin(theta), 0, sin(theta), cos(theta), 0, 0,
cos(PI/2);
tmpRot2 << cos(PI/2), 0, sin(PI/2), 0, 1, 0, -sin(PI/2), 0,

//construct rotation matrix and set result quaternions to match
tmpRot = tmpRot*tmpRot2;
eefQuat = tmpRot;
apprQuat = tmpRot;

//load the EEF pose orientation
res.poseEEF.orientation.x = eefQuat.x();
res.poseEEF.orientation.y = eefQuat.y();
res.poseEEF.orientation.z = eefQuat.z();
res.poseEEF.orientation.w = eefQuat.w();

//load the APPR pose orientation
res.poseAppr.orientation.x = apprQuat.x();
res.poseAppr.orientation.y = apprQuat.y();
res.poseAppr.orientation.z = apprQuat.z();
res.poseAppr.orientation.w = apprQuat.w();

//settings for the marker to represent the EEF pose
//red
mrkEef.pose.position.x = res.poseEEF.position.x;
mrkEef.pose.position.y = res.poseEEF.position.y;
mrkEef.pose.position.z = res.poseEEF.position.z;
mrkEef.pose.orientation.x = res.poseEEF.orientation.x;
mrkEef.pose.orientation.y = res.poseEEF.orientation.y;
mrkEef.pose.orientation.z = res.poseEEF.orientation.z;
mrkEef.pose.orientation.w = res.poseEEF.orientation.w;
mrkEef.color.a = 1.0;      //make visible

//set the transform for EEF coordinate frames
transEef.setOrigin(tf::Vector3(res.poseEEF.position.x,
res.poseEEF.position.y, res.poseEEF.position.z));
transEef.setRotation(tf::Quaternion(res.poseEEF.orientation.x,
res.poseEEF.orientation.y, res.poseEEF.orientation.z, res.poseEEF.orientation.w));
//br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(),
"world", "EEF"));

//set the transform for APPR coordinate frames
transAppr.setOrigin(tf::Vector3(res.poseAppr.position.x,
res.poseAppr.position.y, res.poseAppr.position.z));
transAppr.setRotation(tf::Quaternion(res.poseAppr.orientation.x,
res.poseAppr.orientation.y, res.poseAppr.orientation.z,
res.poseAppr.orientation.w));

//settings for the marker to represent the approach pose
//blue
mrkAppr.pose.position.x = res.poseAppr.position.x;
mrkAppr.pose.position.y = res.poseAppr.position.y;
mrkAppr.pose.position.z = res.poseAppr.position.z;
mrkAppr.pose.orientation.x = res.poseAppr.orientation.x;

```



```

    }// end if loop for errCount==5

    if(errCount==5){
        std::cout << "No valid grasps found\n";
    }

    break;

    default :
        std::cout << "Error: unrecognized object";
        res.success = 0;
        break;

}
// basic structure:
// based on object, generate grasp
// check against IK
// if invalid then modify accordingly and check again etc

return true;
}

//*****
//main function
//this function is the main loop of the grasp planning server node
//it initializes key variables and waits for a request for a grasp plan
//once a request is received the callback function is invoked and results returned
//*****
int main(int argc, char **argv)
{
    ros::init(argc, argv, "grasp_planner_server");
    ros::NodeHandle n;

    ik_client = n.serviceClient<moveit_msgs::GetPositionIK>("compute_ik");

    tf::TransformBroadcaster br;
    //initialize transforms
    transEef.setOrigin(tf::Vector3(0.0, 0.0, 0.0));
    transAppr.setOrigin(tf::Vector3(0.0, 0.0, 0.0));
    transWorld.setOrigin(tf::Vector3(0.0, 0.0, 0.0));
    transEef.setRotation(tf::Quaternion(0, 0, 0, 1));
    transAppr.setRotation(tf::Quaternion(0, 0, 0, 1));
    transWorld.setRotation(tf::Quaternion(0, 0, 0, 1));

    ros::ServiceServer service = n.advertiseService("grasp_plan", grasp_plan_cb);
    ROS_INFO("Ready to grasp plan.");
}

```

```

ros::Rate r(1);
ros::Publisher marker_pub =
n.advertise<visualization_msgs::Marker>("visualization_marker", 1);

marker.pose.position.x = 0;
marker.pose.position.y = 0;
marker.pose.position.z = 0.1;
marker.pose.orientation.x = 1.0;
marker.pose.orientation.y = 1.0;
marker.pose.orientation.z = 0.0;
marker.pose.orientation.w = 1.0;
marker.scale.x = 0.1;
marker.scale.y = 0.1;
marker.scale.z = 0.2;
marker.color.r = 0.0f;
marker.color.b = 0.0f;
marker.color.g = 1.0f;
marker.color.a = 1.0;

mrkEef.pose.position.x = -0.45;
mrkEef.pose.position.y = -0.45;
mrkEef.pose.position.z = 0.7;

mrkEef.scale.x = 0.5;
mrkEef.scale.y = 0.1;
mrkEef.scale.z = 0.1;
mrkEef.color.r = 1.0f;
mrkEef.color.g = 0.0f;
mrkEef.color.b = 0.0f;
mrkEef.color.a = 1.0;

mrkAppr.pose.position.x = -0.65;
mrkAppr.pose.position.y = -0.65;
mrkAppr.pose.position.z = 0.80;
mrkAppr.scale.x = 0.5;
mrkAppr.scale.y = 0.1;
mrkAppr.scale.z = 0.1;
mrkAppr.color.r = 0.0f;
mrkAppr.color.g = 0.0f;
mrkAppr.color.b = 1.0f;
mrkAppr.color.a = 1.0;

while (ros::ok()){
    uint32_t shape = visualization_msgs::Marker::CYLINDER;
    uint32_t arrow = visualization_msgs::Marker::ARROW;

    marker.header.frame_id = "/world";
    mrkEef.header.frame_id = "/world";

```

```

mrkAppr.header.frame_id= "/world";

marker.header.stamp = ros::Time::now();
mrkEef.header.stamp = ros::Time::now();
mrkAppr.header.stamp = ros::Time::now();

marker.ns = "basic_shapes";
mrkEef.ns = "basic_shapes";
mrkAppr.ns = "basic_shapes";

marker.id = 0;
mrkEef.id = 1;
mrkAppr.id = 2;

marker.type = shape;
mrkEef.type = arrow;
mrkAppr.type = arrow;

marker.action = visualization_msgs::Marker::ADD;
mrkEef.action = visualization_msgs::Marker::ADD;
mrkAppr.action = visualization_msgs::Marker::ADD;

marker.lifetime = ros::Duration();
mrkEef.lifetime = ros::Duration();
mrkAppr.lifetime = ros::Duration();

marker_pub.publish(marker);
marker_pub.publish(mrkEef);
marker_pub.publish(mrkAppr);

br.sendTransform(tf::StampedTransform(transEef, ros::Time::now(), "world",
"EEF"));
br.sendTransform(tf::StampedTransform(transAppr, ros::Time::now(), "world",
"APPR"));

ros::spinOnce();
}

return 0;
}

```

## References

- Agile Planet, Inc. "Kinematix Overview." Home. Accessed August 2013.  
<http://www.agileplanet.com/technology>.
- Barrett Technology Inc. BarrettHand Datasheet. 2011.
- Bicchi, A., and V. Kumar. "Robotic Grasping and Contact: a Review." Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on 1(2000): 348-53.
- Bicchi, Antonio. "On the Problem of Decomposing Grasp and Manipulation Forces in Multiple Whole-limb Manipulation." Robotics and Autonomous Systems 13, no. 2 (1994): 127-47.  
doi:10.1016/0921-8890(94)90055-8.
- Brabec, C., K. Schroeder, J. Williams, B. O'Neil, J. Hashem, and M. Pryor. "Reducing the Operator's Burden during Teleoperation Involving Contact Tasks." Proceedings of ANS EPRRS, Knoxville, TN. 2011.
- Chitta, Sachin, Sucan, Ioan, and Pooley, Acorn. MoveIt! - Moving Robots into the Future. Presented at the International Conference on Robotics and Automation(ICRA), May 2013.
- Coldewey, Devin. "New Kinect Can Track You so Well ... You May Not Want It in Your House." NBC News, June 12, 2013. Accessed June 12, 2013.  
<http://www.nbcnews.com/technology/new-kinect-can-track-you-so-well-you-may-not-6C10287970>.
- Cutkosky, M. R., and P. K. Wright. "Friction, Stability and the Design of Robotic Fingers." The International Journal of Robotics Research 5, no. 4 (1986): 20-37.  
doi:10.1177/027836498600500402.
- Cutkosky, M. R., and R. D. Howe. "Human Grasp Choice and Robotic Grasp Analysis." In Dextrous Robot Hands, 5-31. New York: Springer- Verlag, 1990.
- Deb, Satyaranjan, and Sankha Deb. Robotics Technology and Flexible Automation. New Delhi: Tata McGraw-Hill Education, 2010.
- DLR. "Data Sheet of DLR Hand II." DLR. Accessed 2013.  
[http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3802/6102\\_read-8922/](http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-3802/6102_read-8922/).
- Edwards, S.M. and C.L. Lewis. ROS-Industrial — Applying the Robot Operating System (ROS) to Industrial Applications. Presented at the International Conference on Robotics and Automation/Robot Operating System Developer Conference (ICRA/ROSCon), St. Paul, Minnesota, May 2012.



- Elkval, S., and D. Kragic. "Learning and Evaluation of the Approach Vector for Automatic Grasp Generation and Planning." 4715-720. Proceedings of Robotics and Automation, 2007 IEEE International Conference on, Roma. 2007. doi:10.1109/ROBOT.2007.364205.
- Eppner, Clemens, and Oliver Brock. "Grasping Unknown Objects by Exploiting Shape Adaptability and Environmental Constraints." Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013.
- Hanafusa, H. and Asada, H.. "Stable Prehension by a Robot Hand with Elastic Fingers." Paper Presented at the Meeting of the Proceedings of the 7th International Symposium on Industrial Robots, Tokyo, 1977.
- Howe, R. D., and M. R. Cutkosky. "Dynamic Tactile Sensing: Perception of Fine Surface Features with Stress Rate Sensing." IEEE Transactions on Robotics and Automation 9, no. 2 (April 1993): 140-51.
- Huebner, Kai, and Danica Kragic. "Selection of Robot Pre-grasps Using Box-based Shape Approximation." Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on, September 2008, 1765-770.
- Jones, Lynette A., and Susan J. Lederman. Human Hand Function. Oxford: Oxford University Press, 2006.
- Ju, Anne. "Balloon Filled with Ground Coffee Makes Ideal Robotic Gripper | Cornell Chronicle." Balloon Filled with Ground Coffee Makes Ideal Robotic Gripper | Cornell Chronicle. October 25, 2010. <http://www.news.cornell.edu/stories/2010/10/researchers-develop-universal-robotic-gripper>.
- Kapoor, Chetan. A Reusable Operational Software Architecture for Advanced Robotics. Diss., University of Texas at Austin, 1996.
- Markenscoff, X., and C. H. Papadimitriou. "Optimum Grip of a Polygon." The International Journal of Robotics Research 8, no. 2 (1989): 17-29. doi:10.1177/027836498900800202.
- Massa, B., S. Rocella, M. C. Carrozza, and P. Dario. "Design and Development of an Underactuated Prosthetic Hand." Proceedings of Proc. of the IEEE Intl. Conf. on Robotics and Automation, Washington, DC. 2002.
- Meka. H2 Compliant Hand. 2012.
- Mesa Imaging. SR4000 Data Sheet. 2011.
- Microsoft Developer Network (MSDN). "Kinect Sensor." Kinect Sensor. 2012. <http://msdn.microsoft.com/en-us/library/hh438998.aspx>.
- Miller, A.T., Knoop, S., Christensen, H.I., and Allen, P.K. Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference On, 2, Page 1824--1829. IEEE, (2003)
- Modbus Organization. "Modbus FAQ." Modbus. Accessed 2013. <http://www.modbus.org/faq.php>.

- Murakami, Kouji, Kazuya Matsuo, Tsutomu Hasegawa, and Ryo Kurazume. "A Decision Method for Placement of Tactile Elements on a Sensor Glove for the Recognition of Grasp Types." *IEEE/ASME Transactions on Mechatronics*, 2009. doi:10.1109/TMECH.2009.2023647.
- Napier, John Russell. *Hands*. New York: Pantheon Books, 1980.
- Nguyen, V.-D. "Constructing Force- Closure Grasps." *The International Journal of Robotics Research* 7, no. 3 (1988): 3-16. doi:10.1177/027836498800700301.
- Nieuwenhuisen, Matthias, Joerg Stueckler, Alexander Berner, Reinhard Klein, and Sven Behnke. "Shape-Primitive Based Object Recognition and Grasping." *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference On*, May 2012.
- O'Neil, B. E. *Graph-Based World-Model for Robotic Manipulation*. Thesis, University of Texas at Austin, 2010.
- O'Neil, B. E. *Object Recognition and Pose Estimation for Manipulation in Nuclear Materials Handling Applications*. Diss., University of Texas at Austin, 2013.
- Ponce, J., and Faverjon, B. "On Computing Three Finger Forceclosure Grasp of Polygonal Objects." *IEEE Trans. Robotics Automat.*, Vol. 11, N. 6, Pp. 868-881, 1995.
- Ponce, J., S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet. "On Computing Four-Finger Equilibrium and Force-Closure Grasps of Polyhedral Objects." *The International Journal of Robotics Research* 16, no. 1 (1997): 11-35. doi:10.1177/027836499701600102.
- Quigley, M., B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. "ROS: An Open-source Robot Operating System." *Proceedings of Proc. Open-Source Software Workshop of the International Conference on Robotics and Automation (ICRA)*. 2009.
- Rao, Deepak, Quoc V. Le, Thanathorn Phoka, Morgan Quigley, Attawith Sudsang, and Andrew Y. Ng. "Grasping Novel Objects with Depth Segmentation." 2578-585. *Proceedings of Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Taipei. 2010. doi:10.1109/IROS.2010.5650493.
- Rao, Kashipati, Gerard Medioni, Huan Liu, and George A. Bekey. "Shape Description and Grasping for Robot Hand-eye Coordination." *Control Systems Magazine, IEEE* 9, no. 2 (February 1989): 22-29.
- Richtsfeld, A., T. Morwald, J. Prankl, M. Zillich, and M. Vincze. "Segmentation of Unknown Objects in Indoor Environments." 4791-796. *Proceedings of Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Vilamoura. 2012. doi:10.1109/IROS.2012.6385661.
- Robotiq. *Robotiq 2-Finger Model - 200 Specifications*. 2013. <http://robotiq.com/media/Robotiq-2-Finger-Adaptive-Gripper-200-Specifications.pdf>.

- Robotiq. Robotiq 3-Finger Adaptive Robot Gripper Instruction Manual. 2011. <http://robotiq.com/media/ROBOTIQ-INSTRUCTIONMANUAL-S-MODEL-ROBOT-GRIPPER.pdf>.
- Robotiq. Robotiq Adaptive Gripper 3-Finger Model Specification Sheet. 2013. <http://robotiq.com/media/Robotiq-3-Finger-Adaptive-Robot-Gripper-Specifications.pdf>.
- ROS.org. "ROS." ROS.org. 2013. <http://www.ros.org/wiki/ROS>.
- Salisbury, J. K., and J. J. Craig. "Articulated Hands: Force Control and Kinematic Issues." *The International Journal of Robotics Research* 1, no. 1 (1982): 4-17. doi:10.1177/027836498200100102.
- Salisbury, J. K. "Kinematics and Force Analysis of Articulated Hands." Thesis, Stanford University, 1982.
- Saxena, A., J. Driemeyer, and A. Y. Ng. "Robotic Grasping of Novel Objects Using Vision." *The International Journal of Robotics Research* 27, no. 2 (2008): 157-73. doi:10.1177/0278364907087172.
- Schroeder, Kyle. Diss., University of Texas at Austin, 2013.
- SCK CEN. LHMA Hot Cell. Accessed 2013. <http://www.sckcen.be/en/Media/Images/Our-Research/Facilities/LHMA-hot-cell>.
- Shadow Robot Company. Shadow Dexterous Hand Technical Specification. 2013.
- Shastri, S. V., and Thea Iberall. *Dextrous Robot Hands*. New York: Springer-Verlag, 1990.
- Syntouch LLC. BioTac Product Manual. 2013.
- Thompson, A. W. Nuclear Residues Repacking Glovebox, Building 440, Rocky Flats Nuclear Weapons Plant. 2002.
- Trinkle, J.C. On the Stability and Instantaneous Velocity of Grasped Frictionless Objects. *IEEE Trans. on Robotics and Automation*, Vol.8, No.5, 1992.
- Varadarajan, K. M. "Object Part Segmentation and Classification in Range Images for Grasping." 21-27. *Proceedings of Advanced Robotics (ICAR), 2011 15th International Conference on*, Tallinn. 2011. doi:10.1109/ICAR.2011.6088647.
- Wolf, Andreas, Ralf Steinmann, and Henrik Schunk. *Grippers in Motion: The Fascination of Automated Handling Tasks*. Berlin: Springer, 2005.
- Yamanobe, Natsuki, and Kazuyuki Nagata. "Grasp Planning for Everyday Objects Based on Primitive Shape Representation for Parallel Jaw Grippers." *Robotics and Biomimetics (ROBIO)*, 2010 IEEE International Conference On, December 2010, 1565-570.
- Yaskawa America, Inc. SIA5D Datasheet. 2012. Accessed 2013. <http://www.motoman.com/datasheets/SIA5D.pdf>.